Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Vysoce dostupný SIP server s rychlým failover mechanizmem

High Available SIP Server with Fast Failover Mechanism

2018 **Bc. Daniel Mašík**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Daniel Mašík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Vysoce dostupný SIP server s rychlým failover mechanizmem
High Available SIP Server with Fast Failover Mechanism

Jazyk vypracování: čeština

Zásady pro vypracování:

V ak. roce 2009/2010 byla odevzdána diplomová práce [1], ve které bylo pro dosažení HA ve VoIP využito databázové replikace v kombinaci s VRRP, Keepalived a Heartbead mechanizmů, což bylo aktuální v době formování zadání práce (2008). Mezitím ovšem nastal velký posun v oblasti virtualizace a otevřely se nové možnosti, což je i motivace pro toto zadání.

Článek kolegů z ČVUT [2] se zabývá možností živé replikace virtualizovaných VoIP serverů v cloudu s využitím XEN hypervizoru, hlavní autor dále rozpracoval spolehlivostní model pro zvýšení odolnosti VoIP řešení proti výpadku ve své dizertační práci [3]. Další zajímavá práce se zabývá síťovou virtualizací a možnostmi jejího využití pro HA VoIP řešení v rozsáhlých sítích [4].

Tato diplomová práce si klade za cíl praktické nasazení vhodného řešení vysoce dostupného SIP serveru v cloudu a rigorózní evaluaci jeho chování při výpadku primárního prvku.

1. Využití VoIP pro hlasové služby, jejich dostupnost a vývoj v ČR.
2. Virtualizace, cloudová řešení a failover mechanizmy.
3. Návrh a praktická realizace HA SIP serveru.
4. Evaluace provozních vlastností a chování během výpadku.
5. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] Travnicek, L. *Komunikační server s vysokou dostupností*. VŠB-TU Ostrava, Diplomová práce, 2010.
[2] Hlavacek, J., Bestak, R. *Live Replication of Virtualized VoIP Servers*. In The Eighth International MultiConference on Computing in the Global Information Technology. Silicon Valley: International Academy, Research and Industry Association, 2013, p. 277-282.
[3] Hlavacek, J. Robustness of VoIP systems. FEL ČVUT v Praze, Dizertační práce, 2015.
[4] Septama, H.D., Ulvan, A., Bestak, R., Hlavacek, J. *High available VoIP server failover mechanism in Wide Area Network*. Telecommunication Computing Electronics and Control, Volume 13, Issue 2, 2015, pp. 739-744.

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *30. dubna 2018*

……
podpis studenta

Poděkování

Rád bych poděkoval prof. Ing. Miroslavu Vozňákovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské/diplomové práce.

Abstrakt

Podmětem pro tuto diplomovou práci bylo nalézt optimální řešení k zajištění vysoké dostupnosti pro služby VoIP (voice over IP), neboli přenos hlasu (multimediálního obsahu) přes datovou síť pomocí IP (internet protokolu). Aktuálně lze nalézt více možností, jak vysoké dostupnosti dosáhnout. Mezi hlavní problémy se řadí výměna IP adresy a replikace dat potřebných k činnosti. Zvolené řešení replikace dat, vysoká dostupnost kontejnerů a živé migrace virtuálních strojů byly testovány a poté zohledněny ve výsledcích, které porovnávají rozdíly mezi těmito řešeními.

Klíčová slova

Telekomunikace; replikace; cluster; VoIP; vysoká dostupnost; Asterisk; virtualizace; Docker

Abstract

The subject for this thesis was find an optimal solution for the system to ensure high availability for VoIP (voice over IP), which is the transmission of voice (multimedia content) over the data network using IP (Internet protocol). Actually, there are more ways how high availability create. Major issues include exchange of IP addresses and replication of data. The data replication solution, the high availability of containers, and the virtual migration of virtual machines were tested and results compared.

Key words

Telecommunication; replication; cluster; VoIP; high availability; Asterisk; virtualization; Docker

Seznam použitých zkratek

Zkratka	Význam
ARP	Address Resolution Protocol
ASHA	Asterisk High Availability
CRM	Cluster Resource Management
CRMSH	Cluster Management Shell
DRBD	Distributed Replicated Block Device
GNU	General Public Licence
HA	High Available
IETF	Internet Engineering Task Force
IP	Internet Protokol
ISDN	Integrated Services Digital Network
KVM	Kernel-based Virtual Machine
MCU	Multipoint Control Unit
NAT	Network Address Translation
NFS	Network File Systém
NFV	Network function virtualization
NTP	Network Time Protocol
OSD	Object storage deamon
PSTN	Public switched telephone network
QEMU	Quick Emulator
RADOS	Reliable Autonomic Distributed Object Store
RAID	Redundant Array of Independent Disks
RDMA	Remote direct memory access
RTP	Real-Time Transmission Protocol
SIP	Session Initiation Protocol
SMB	Server Message Block
SSH	Secure Shell
TCP	Transmission Control Protocol

TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
VM	Virtual machine
VoIP	Voice over IP
VRRP	Virtual Router Redundancy Protocol
ZTF	Zettabyte File Systém

Seznam použitých termínů

Termín	Význam termínu
Load balancing	Rozložení zatížení mezi dva nebo více počítačů, popřípadě jiných zařízení, aby bylo dosaženo optimálního využití.
Framework	Softwarová struktura, která slouží jako podpora při programování a vývoji.
Cluster	Seskupení volně vázaných počítačů, které spolu úzce spolupracují, takže se navenek mohou tvářit jako jeden počítač.
Uzel	Člen clusteru.
Open Source	Software s otevřeným zdrojovým kódem.
Deamon	Označení programu, který je spuštěn dlouhodobě a není v přímém kontaktu s uživatelem.
Failover	Označení pro pojištění v případě výpadku, kdy dochází k přepnutí na záložní systém.

Obsah

Úvod

S rostoucí popularitou datových sítí sílí i popularita VoIP. Takřka vytlačila starší technologii telefonní sítě, dnes v hojném počtu využívanou převážně pro přenos dat. Úkolem mé diplomové práce bylo zvýšit dostupnost serverů VoIP za použití aktuálních volně dostupných technologií a navázat tím na diplomovou práci pana Trávnička (2010) a dizertační práci pana Hlaváčka (2013).

V první kapitole se stručně věnuji technologiím pro vysokou dostupnost využitých panem Hlaváčkem a panem Trávníčkem. Dále popisuji kritéria a vlastnosti, které by měly prvky VoIP pro vysokou dostupnost splňovat.

V následující kapitole rozebírám jednotlivá řešení vysoké dostupnosti, jež mohou být v dnešní době použita. Jsou popsány jednak technologie komerční implementující všechny prvky vysoké dostupnosti do jednoho kompletního systému, tak i technologie volně dostupné, které jsou ovšem nutné ve většině případů kombinovat mezi sebou, aby byla zaručena vysoká dostupnost.

Ve třetí kapitole je vysvětlena samostatná technologie VoIP, zejména SIP. Zaměřuji se převážně na jejich vlastnosti a možnosti, které nabízejí. Pátá kapitola navazuje na předchozí kapitolu, jelikož je zde popsána ústředna Asterisk, jež byla v této práci použita. Stručně charakterizuji její vlastnosti a možnosti nastavení.

Vedle ústředny Asterisk je rozebrána technologie virtualizace, protože byla využita pro testování a také u jednoho z řešení. Zabývám se typy a možnostmi, co vše lze virtualizovat. Šestá kapitola se zaobírá technologií kontejnerů, které jsou nyní na vzestupu. Zejména systém Docker, jenž s kontejnery pracuje, nabízí technologii pro vysokou dostupnost a vyrovnání zátěže. Je tedy použit ve druhém řešení.

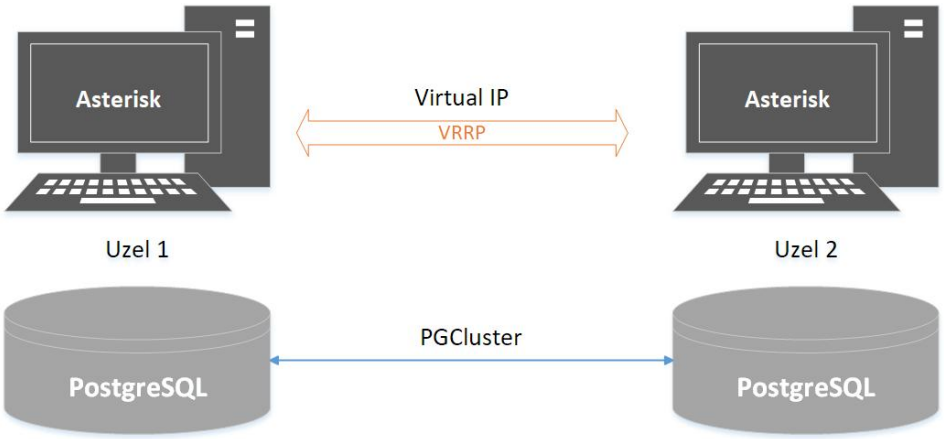
V další kapitole jsem poté vybral celkem tři možnosti, jak zajistit vysokou dostupnost s rozdílnými typy technologií. Popisuji jednotlivé technologie, jejich instalaci, konfiguraci a následnou implementaci s telefonní ústřednou Asterisk.

Na závěr pak otestuji všechny funkční řešení pomocí softwaru SIPp, který analyzuje protokol SIP s pomocí definovaných scénářů. Z celkových naměřených hodnot jsou poté vyhotoveny grafy, které porovnávají jednotlivá řešení.

1 Vysoká dostupnost

V této kapitole objasním základní vlastnosti, které by měl systém pro vysokou spolehlivost splňovat. Vysoká dostupnost se obvykle dosahuje prostřednictvím clusteru, což znamená, že dva, čí více strojů jednají jako jeden pro určitý účel. Existuje mnoho způsobů, jak vytvořit cluster, každý s vlastními výhodami, riziky, náklady a kompromisy. Jsou ovšem vlastnosti, které by měl systém HA splňovat.

Inspirací pro tuto práci byla diplomová práce pana Trávnička (2010) a dizertační práce pana Hlaváčka (2013). V práci pana Trávnička je vysoká dostupnost zajištěna migrací IP adresy v případě selhání odesílatele, při sledování stavu SIP serveru. Z této úvahy lze vyvodit, že zvýšením počtu serverů můžeme docílit vetší dostupnosti sítě VoIP. Pro migraci IP adres byl využit VRRP (virtual Router Redundancy Protocol). V rámci VRRP byl na dvou stanicích naistalován daemon, který nastaví sdílenou virtuální IP adresu mezi stanicemi. Jeden ze serverů je aktivní, označen jako hlavní a používá virtuální IP adresu. Druhý je v pohotovostním režimu a převezme funkci, pokud hlavní server selže. Ovšem nevýhodou tohoto protokolu je škálovatelnost, funguje pouze pro dva servery. Není tedy vhodný pro zajištění vetší stability, při použití více serverů. Základní funkci výměny IP adresy zajišťuje deamon heartbead, který spouští služby podle nastavení v konfiguraci. Od verze 2 podporuje i spojení více než dvou počítačů do clusteru. Verze ústředny Asterisk, která byla použita v této práci, využívala zastaralou databázi, která nevyhovovala vysoké dostupnosti serveru. Proto byla nahrazena databází Postgresql, která nabízí větší konfigurační možnosti. Pro zajištění stability musel být použit nástroj PGCluster replicator. Nástroj provádí zápisy do databáze na ostatní záložní databáze. Což snižuje potřebné přenosové pásmo. Při obnově nebo přidání nového databázového serveru se používá program Rsync, který kopíruje adresář obsahující data databáze, z již funkčního serveru na nový server, je schopen filtrovat neexitující data a ty poté překopírovat na nový server, viz. [1]. Nejsou tak zbytečně přenášena existující data. Celé schéma je znázorněno v grafickém schématu (Obrázek 1.1).



Obrázek 1.1: *Schéma zapojení - p. Trávníček*

Práce pana Hlaváčka byla zaměřena na celkovou vysokou dostupnost, nikoliv jen u VoIP. Nastínila dvě možná řešení pro zajištění vysoké dostupnosti SIP serveru. První řešení bylo použití virtuálních strojů s možností replikace. Zahrnuta byla i replikace v reálném čase a její vliv na aplikaci. Virtualizace je široce využívána pro své výhody, jako je lepší využití zdrojů, flexibilita a škálovatelnost. Kromě toho může být také použita ke zlepšení dostupnosti systému pomocí techniky živé migrace. Hlavní výhoda virtualizace spočívá v migraci, která je pro aplikace zcela transparentní. Výzvou mechanismu živé migrace je zajistit konzistenci dat v případě selhání primárního serveru. Kvůli minimalizaci přerušení a uchování výkonů se replikace provádí nepřetržitě. Jakmile je zjištěna chyba, replikované zařízení se obnoví na záložním serveru bez přerušení hovoru. K oznámení nového umístění virtuální IP adresy se použije bezplatná žádost o ARP (Address Resolution Protocol). Výsledky této práce ovšem ukazují, že předchozí implementace kontinuální replikace není vhodná pro data v reálném čase. Vytváří závažný rozptyl zpoždění, a tím zhoršuje kvalitu hovoru. Proto provedl úpravy na hypervisoru virtualizace. Celý stav počítače byl replikován, všechny volání s přidruženými kontexty byly zachovány, pokud došlo k selhání. Toto řešení bylo vhodné pro systémy VoIP, viz. [2].

Další návrh byl založen na barevných stochastických Petriho sítích s logaritmicko-normálním rozložením četností poruch. Využíval standardních programovacích technik pro zajištění dostupnosti a odolnosti vůči chybám. Řešení je ovšem vyhovující pro návrh nové aplikace, nikoliv pro ty stávající. Taktéž celý návrh je velice složitý, což může mít za následek veliká úskalí.

1.1 Kritéria pro vysokou dostupnost

1.1.1 Spolupráce a autonomie

Tato kritéria jsou velice důležitá při navrhování a vytváření serveru s vysokou dostupností. Aby řešení bylo stabilní, musí být automatické a autonomní. Některá řešení HA

zahrnují sdílení hardwaru, softwaru, logického zařízení atd. Problémem s tímto přístupem je ten, že se vytvoří jediný bod selhání. Pokud například cluster sdílí propojení hardwarového kanálu (např. propojení dvou počítačů prostřednictvím ethernet kabelů), pak v případě selhání kanálu, selže celý cluster. Ve skutečném clusteru musí mít uzly spojení autonomní; tj. že nesdílí žádný hardware, software, logická zařízení atd., viz [11].

1.1.2 Synchronizace a škálovatelnost dat

K tomu, aby cluster zůstal užitečný a stabilní, je nutné udržet údaje synchronizované. V případě selhání pak jeden uzel začne v bodě, kde druhý selhal. Synchronizace je dalším důležitým aspektem při navrhování HA clusterů a jedná se o jeden z největších problémů.

Jistým řešením tohoto problému může být sdílení disků čili sdílení údajů a dat. Zde však nastává problém, v případě poškození jednoho disku v uzlu se poškodí disk z uzlu druhého. Složitějším přístupen je synchronizace dat mezi uzly. Existuje celá řada nástrojů, které pomohou synchronizovat data, nicméně je důležité, aby synchronizace probíhala pouze v případě, kdy jsou uzly rovnocenné (jinak riskujeme synchronizaci poškozených dat, jako u předchozího příkladu).

Následující technologie synchronizují data mezi uzly. Buď tento proces probíhá pod kontrolou softwaru HA, nebo samostatně. Například kopírování souborů, rsync, nebo klonování údajů v databázi. Jednou z klíčových výhod těchto technologií je, že kdyby jedna z nich selhala, pak lze pozastavit jakoukoliv synchronizaci, aby se zabránilo poškození jiného uzlu.

1.1.3 Sdílení dat

Následující technologie sdílejí logický/fyzický disk: DRBD, iSCSI, NFS, SAMBA. Výhodou této technologie je jednoduchost, možnost použití otevřených softwarů. Některé sdílené datové technologie jako DRBD vytvářejí lokální vyrovnávací paměť, která může mít za následek syndrom *Split Brain*. Ten může nastat po obnovení spojení, kdy hostitel nemůže zjistit správnou kopii dat. Všechny tyto technologie také selhávají v širokopásmových sítích, spotřebovávají totiž enormní šířku pásma.

1.1.4 Detekce poruchy

K tomu, aby cluster mohl zůstat produktivní, musí vědět, kdy je uzel v pořádku a kdy selhal. Je nutné, aby každé řešení monitorovalo a zjišťovalo stupně zhoršení a inteligentně určilo, co se nezdařilo. Mezi jednotlivé poruchy můžeme například zařadit:

- vypnutí stroje,
- velhání operačního systému,
- selhání hovoru,
- selhání sítě.

Pro toto řešení mohou být použity open source balíčky *Heartbeat*, které detekují, zda je proces v pořádku.

1.1.5 Fyzické oddělení uzlů

Ačkoliv toto kritérium není až tak důležité pro malé společnosti, je nutné jej brát v potaz. Důležitým aspektem je, aby jednotlivé lokality byly od sebe geograficky odděleny významnou vzdáleností. Přínosem fyzického oddělení je, že katastrofy, které narazí na jedno datové centrum, město, region, pravděpodobně neovlivní druhou lokalitu.

1.1.6 Šifrování

Další úvahou je šifrování dat z důvodu ochrany uzlů před napadením a útoky *man in the middle*. Jakákoliv data, která musí cestovat přes internetovou síť, by měla být šifrována. Například technologie NFS, či iSCSI nešifrují provoz.

1.1.7 Odolnost proti výpadku

Systém HA by měl mít více kombinací odolnosti proti výpadkům. Například mít redundantní síťová rozhraní, napájecí zdroje a disky. To umožní, aby nejběžnější selhání proběhlo transparentně uživatelům bez výpadku. O toto se dnes umí postarat hardwarové prvky, které jsou k tomu určeny jako záložní napájecí zdroje.

1.1.8 Vyrovnání zatížení

Pro veliké telefonní ústředny je potřebné správně rozvržené zatížení, aby nedošlo k přetížení ústředny, a tím ke kolapsu systému. V dřívějších dobách byla nezbytná zvladatelnost velkých objemů hovorů. Ovšem dnes i s nízkými náklady severy dokáží obsloužit až stovku hovorů současně a vyrovnání zatížení je zřídka implementováno, viz [11].

2 Řešení vysoké dostupnosti

Metody jak zajistit HA je mnoho. Lze je vytvořit jak softwarově, tak i hardwarově. Dále můžeme řešení rozdělit na komerční a volně dostupné open source. Celá práce je zaměřena spíše na řešení pomocí softwaru. Proto v této kapitole popíši nejvíce zmiňovaná softwarová řešení. Budu se zabývat řešením komerčním i open source.

2.1 Komerční řešení

Výhodou těchto řešení je úplná nezávislost uzlů, rozsáhlá detekce a všechny funkce v jednom integrovaném řešení (synchronizace dat, detekce poruch, sdílení IP atd.). Není nutné spojovat více služeb dohromady, aby byla splněna HA. Mnoho z těchto řešení využívá služeb cloudu, kdy se koncový zákazník pouze připojí a autorizuje. Všechny technické záležitosti spadají pod dodavatele daných služeb.

2.1.1 Bicom Systems SERVERware

Virtualizační platforma Linux určená pro telefonován poskytuje širokou škálu služeb IP a aplikací. Zaručuje vysokou dostupnost a odolnost vůči chybám. Poskytuje komplexní management, flexibilitu, obnovu a neomezenou škálovatelnost hostitele a virtuálních serverů. Nabízí dvě edice:

Edice server: Tato edice má dva módy nastavení. *Non-Redundant* mód, který lze nasadit s lokální redundancí hardwaru pro zajištění vysoké dostupnosti, pokud primární server selže. Využívá zrcadlení serveru a poskytuje rychlé převzetí služeb při selhání. Poté *High availability mód*, který využívá ISDN společnosti Junghann v případě, že selže nebo ztratí napájení.

Edice network: Tato edice obsahuje až 256 hostitelských serverů vytvářející farmu virtuálních privátních serverů, kterým je nabízena platforma pro poskytování IP služeb. To dovoluje nabídnout redundantní, flexibilní a škálovatelné služby IP, jako je pošta, web, hostované pobočkové ústředny a jiné, viz [10].

2.1.2 Thirdlane Elastic Cloud PBX

Jedná se softwarovou platformu umožňující poskytovatelům služeb nasadit rozsáhlé obchodní komunikační služby a služby pro spolupráci v cloudu pomocí více fyzických, nebo virtuálních serverů pro optimální výkon a škálování. Vysoká dostupnost, redundance a geo-redundance jsou jádrem tohoto softwaru. ThirdCloud Cluster umožňuje prakticky neomezenou škálovatelnost, kde mohou být přidány a konfigurovány více uzlů sestávajících se z telefonních serverů, komunikačních serverů a dalších komponent, které poskytují jednu velkou platformu. Integruje Kamailio SIP Server a Asterisk.

2.1.3 HAAst

Software HAAst (High Availability for Asterisk) nabízí rychlou automatickou kontrolu nad selhanými uzly, celkovou autonomii uzlů, sdílení IP, pokročilé zjišťování stavu uzlů,

inteligentní synchronizace souborů a databází apod. Vytváří cluster s vysokou dostupností z jakékoliv dvojice serverů Asterisk. HAAst dokáže rozpoznat řadu poruch na jednom serveru Asterisk a automaticky přenést řízení na druhý server, což má za následek telefonní prostředí s minimálním prostojem. Integrovaná síťová kontrola umožňuje sdílení jedné IP adresy mezi servery, takže klient či telefon se automaticky připojují k aktivnímu serveru Asterisk bez změny. HAAst zahrnuje škálu snímačů pro detekci selhání serveru, jeho hardwaru, sítě atd. To vše tak přispívá k celkové stabilitě.

2.1.4 **FreePBX HA**

FreePBX HA je komerčně vyvinuté řešení s vysokou dostupností, které přepracovalo platformu FreePBX k integraci DRBD, správce clusterů a Pacemaker. To umožňuje automatické zrcadlení a přepnutí mezi dvěma systémy FreePBX. Telefony a zařízení jsou registrovány na plovoucí IP adresu, takže převzetí služeb při selhání mezi systémy bude pro ně transparentní. SIP trunky se zaregistrují do aktivního uzlu, a pokud jsou používány PSTN Failover Appliance (volitelné), linky T1 nebo Analog budou směrovány do aktivního uzlu. Poté, když je primární ústředna opravena nebo obnovena, dochází během několika sekund k přepnutí zpět na primární uzel FreePBX. Nastavení a konfigurace FreePBX HA se provádí v grafickém rozhraní FreePBX, které je používáno také jako nástroj pro správu online pro snadné přepínání mezi uzly. Doporučuje se (ale není nutné) instalaci distribuce Asterisk 11 při spuštění FreePBX High Availability, viz [14].

2.1.5 **Elastix HA**

Elastix High Availability je řešení, které snadno umožňuje nasazení clusteru mezi dvěma servery Elastix. Na rozdíl od zálohování z cloudu zaručuje Elastix High Availability provozní kontinuitu systému. V případě selhání hlavního serveru je průměrná doba spuštění serveru slave od tří do čtyř sekund. V současné době je aplikace určena pro dva uzly a v budoucnu se tato funkce zvýší. Všechny konfigurace v prvním uzlu a uzlu druhém se provádějí automaticky, viz [15].

2.1.6 **SARK-HA**

Společnost SARKHA byla původně vytvořena pro zákazníky v Jižní Africe a od té doby se ukázala být velmi populární u uživatelů, jež kladou vysokou hodnotu na dostupnost své pobočkové ústředny. Cílem bylo a je poskytnout pobočkovou ústřednu Asterisk s téměř 100 % dostupností. SARK HA verze se k tomuto cíli dostává velmi blízko. V rámci vývoje SARK verze 3 je sarkha rozšířen přidáním nového modulu nazvaného ASHA (Asterisk High Availability), který je kromě jiného mnohem sofistikovanější a používá rsync přes SSH místo původní metody, sdílení SMB. SHARK HA pracuje s využitím standardní technologie clusteru Linux HA pro provoz dvou serverů Asterisk. Jeden aktivní nebo primární server a jeden pasivní nebo pohotovostní server. V případě selhání Asterisku nebo Linuxu na aktivním serveru bude pohotovostní server automaticky přebírat kontrolu nad všemi prostředky a pokračovat ve službě až do okamžiku, kdy je primární stanice opět připravena převzít odpovědnost. Čas potřebný k

selhání se může lišit v závislosti na tom, jak jsou nastaveny parametry Heartbeat, ale je celkem běžné, že velké systémy (250 až 300 telefonů) mohou selhat po méně než 30 sekundách. Spravuje také synchronizaci dat mezi uzly a přidává některé doplňkové funkce, jako je démon watchdog Asterisk, viz [16].

2.1.7 VMware ESXi

Jedná se o systém vyvinutý společností VMware pro nasazení a obsluhu virtuálních počítačů. Jako hypervisor typ-1, ESXi není softwarová aplikace, která je nainstalována v operačním systému (OS), ale místo toho obsahuje a integruje důležité komponenty operačního systému, jako například jádro. Hypervisor je založen na operačním systému VM kernel. ESXi je exkluzivní hypervisor pro licence VMware vSphere 5.x. Informace o stavu lze nahrát z uloženého konfiguračního souboru. VMkernel ESXi je přímo propojen s agenti VMware a schválenými moduly třetích stran. Správci virtualizace mohou nakonfigurovat VMware ESXi prostřednictvím konzoly nebo VMware vSphere Client. Systém je podporován pouze na hardwaru schváleným firmou VMware.

2.1.8 VMware VShere

VSphere HA je další z nástrojů firmy VMware, který poskytuje vysokou dostupnost pro virtuální stroje spojením virtuálních počítačů a hostitelů do clusteru. Hostitelé v clusteru jsou sledováni a v případě selhání jsou virtuální počítače na neúspěšném hostiteli restartovány na alternativních hostitelích.

Při vytváření clusteru vSphere HA se jako hlavní hostitel automaticky zvolí jeden hostitel. VMware HA podporuje snadnou konfiguraci a monitorování pomocí aplikace Virtual Center. Systém HA zajišťuje, že je k dispozici vždy kapacita (v mezích stanovené kapacity pro převzetí služeb při selhání), aby se restartovaly všechny virtuální počítače postižené selháním serveru (na základě rezervací prostředků konfigurovaných pro virtuální počítače). Hlavní hostitel komunikuje s VirtualCenter Serverem a sleduje stav všech chráněných virtuálních počítačů a uzlů typu slave. Různé typy selhání hostitele jsou možné. Hlavní hostitel musí zjistit a vhodně řešit selhání. Musí také rozlišovat mezi neúspěšným hostitelem a hostitelem, který je v síťovém oddílu nebo se stal izolovaným v síti, viz [23].

2.2 Open source řešení

Velikou výhodou těchto řešení se stává cena, avšak k vytvoření HA je v mnoha případech nutné spojení více těchto služeb dohromady. Může se také vyskytnout problém s podporovou a dokumentací, která není vždy plnohodnotná, a řešení nesplní plnou funkcionalitu, jaká je očekávána. Technologií, kterých lze použít pro HA, je veliké množství. Existuje více softwaru, který vykonává stejnou funkcionalitu, v mnoha případech se od sebe neliší.

2.2.1 Keepalived

Keepalived je směrovací software napsaný v jazyce C. Hlavním cílem tohoto projektu je poskytnout jednoduché a robustní zařízení pro vyvažování zátěže a vysokou dostupnost pro

infrastrukturu systému Linux. Loadbalancing framework spočívá na dobře známém a široce používaném modulu jádra Linux Virtual Server zajišťující vyrovnávání zatížení čtvrté vrstvy. Společnost Keepalived implementuje sadu kontrolních prvků, které dynamicky a adaptivně udržují a spravují vyrovnaný serverový fond podle svého zdraví. Na druhé straně vysoká dostupnost se dosahuje protokolem VRRP. Ten je základním prvkem pro selhání routeru. Keepalived je svobodný software, může být rozšířen, nebo upraven podle podmínek GNU (General Public License).

Obvykle protokol VRRP zajišťuje, že jeden ze zúčastněných uzlů je označen jako hlavní. Záložní uzel naslouchá paketům vysílaných z uzlu s vyšší prioritou. Pokud záložní uzel nebude přijímat packety po dohodnutou dobu, záložní uzel převezme hlavní roli a přebere na sebe nakonfigurované IP adresy. V případě, že existuje více než jeden záložní uzel se stejnou prioritou, vyhrává ten s nejvyšší IP. Výhoda tohoto softwaru je jednoduchost. Umožňuje vytvářet kontrolní skripty. Kontrolní skript znamená skript napsaný v jazyce podle vašeho výběru, který se provádí pravidelně. Ovšem má i své nedostatky. Pokud se totiž dva zúčastněné uzly navzájem nevidí, oba budou mít hlavní stav a oba budou mít stejné IP adresy.

2.2.2 Pacemaker

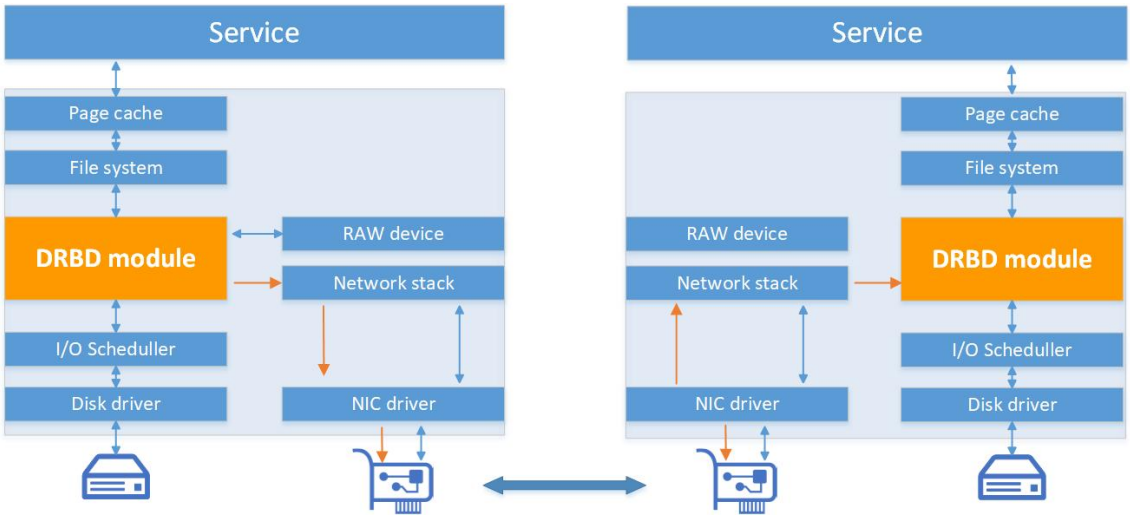
Pacemaker je správce zdrojů clusteru s vysokou dostupností. Jedná se o open source implementací CRM (Cluster Resource Management). Pomocí sady démonu zajišťuje spouštění či naopak zastavení konkrétních služeb (zdrojů). Dosahuje maximální dostupnosti služeb zjištěním a obnovením selhání na úrovni uzlů a zdrojů. K tomu využije možnosti zasílání zpráv a komunikací pomocí Corosync. V podstatě Corosync umožňuje serverům komunikovat jako cluster, zatímco Pacemaker poskytuje schopnost řídit chování clusteru. Mezi hlavní výhody patří:

- detekce a obnovení při selhání stroje a aplikace,
- podporuje prakticky libovolnou konfiguraci redundance,
- podporuje jak kvótové, tak zdroje orientované klastry,
- obsahuje konfigurovatelné strategie pro řešení ztráty uzlu,
- podporuje spouštění a vypínání aplikací bez ohledu na to, zda jsou aplikace zapnuté,
- podporuje aplikace, které musí / nemusí být spuštěny na stejném počítači,
- podporuje aplikace, které musí být aktivní na více strojích,
- podporuje aplikace s více režimy (např. Master / slave),
- prokazatelně správná odezva na případné selhání nebo stav klastru,
- odpověď clusteru na jakékoli podněty lze testovat off-line předtím, než daný stav existuje.

To může udělat pro clustery prakticky jakékoliv velikosti a přichází s výkonným modelem závislostí, který umožňuje správci přesně vyjadřovat vztahy (jak uspořádání, tak umístění) mezi prostředky clusteru. Prakticky vše, co může být napsáno, může být spravováno jako součást pacemakeru. Konfigurace se prování pomocí CRMSH (Cluster Management Shell) příkazů, což přináší určitou složitost.

2.2.3 DRBD

DRBD je distribuovaný replikovaný úložný systém pro platformu Linux. Umožňuje sdílení a výměnu uložených dat díky síťovému zrcadlení. DRBD se může replikovat ve více síťových protokolech a v (aktuálně) třech režimech, od synchronních pro místní clustery HA až po asynchronní pro posun dat na místo obnovy po havárii. Pokud některý z uzlů úložiště v replikovaném prostředí selže, má systém další blokové zařízení, na které se může spoléhat a může bezpečně převzít službu failover. Stručně řečeno, lze jej považovat za implementaci zrcadlení RAID1 s využitím kombinace lokálního disku a jednoho na vzdáleném uzlu, ale s lepší integrací s clusterovým softwarem, jako je heardbead.



Obrázek 2.2: *Schéma architektury DRBD*

Úroveň spojování dat závisí na zvoleném protokolu. DRBD podporuje tři protokoly:

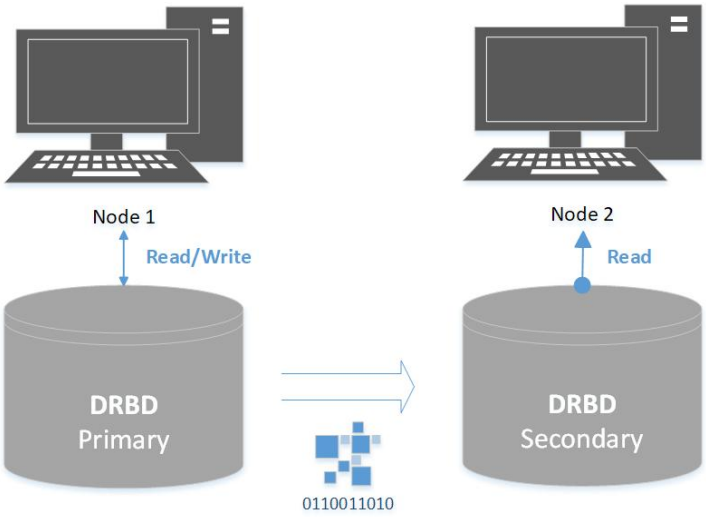
- **Protokol A:** Zápisy se považují za dokončené, jakmile jsou zapsány lokální disky a datový paket byl umístěn do fronty pro odesílání. V případě selhání uzlu může dojít ke ztrátě dat, protože data, která mají být zapsána na disk vzdáleného uzlu, mohou být stále ve frontě k odesílání. Údaje o uzlu převzetí služeb při selhání jsou však konzistentní, ale nejsou aktuální. To se obvykle používá pro geograficky oddělené uzly.
- **Protokol B:** Zapsání na primárním uzlu se považuje za dokončené, jakmile se dokončí zápis lokálního disku a replikační paket dosáhne uzlu. Ke ztrátě dat může dojít v případě souběžného selhání obou účastnických uzlů, protože údaje při přenosu nemusí být zálohovány.

- **Protokol C:** Zápisy jsou považovány za dokončené pouze, až když disky lokálního i vzdáleného uzlu potvrdily úspěšný zápis. Neexistuje ztráta dat, takže se jedná o populární schéma pro seskupené uzly, avšak propustnost je závislá na šířce pásma sítě.

DRBD klasifikuje uzly clusteru jako primární nebo sekundární. Primární uzly mohou iniciovat modifikace nebo zapisovat, zatímco sekundární uzly nemohou. To znamená, že sekundární DRBD uzel neposkytuje přístup a nemůže být připojen. Dokonce i přístup jen pro čtení je z důvodů koherence mezipaměti zakázán. Sekundární uzel je přítomen především v případě chyby jako zařízení při selhání. Sekundární uzel se může stát primárním v závislosti na konfiguraci sítě. Přiřazení a určování rolí provádí software pro správu clusterů.

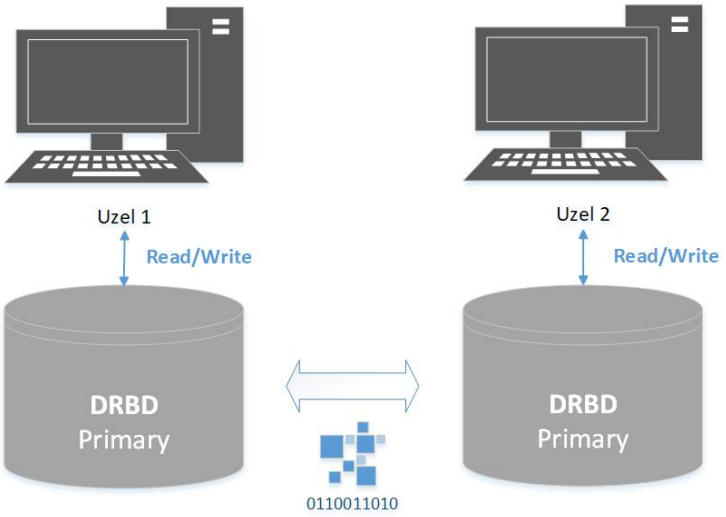
Existují různé způsoby, kterými může být uzel označen jako primární:

Single Primary: Primární označení je dáno jednomu členu klastru. Protože pouze jeden člen clusteru zpracovává data, je tento režim užitečný u běžných souborových systémů, jako je ext3 nebo XFS.



Obrázek 2.3: *Schéma DRBD v režimu Single Primary*

Dual Primary: Oba clusterové uzly mohou být primární a mohou měnit data. To se obvykle používá v souborových systémech clusterů, jako je například ocfs2.



Obrázek 2.4: *Schéma DRBD v režimu Dual Primary*

V případě dual primary mohou být oba uzly zapsány do stejného sektoru disku, čímž jsou data nekonzistentní. Aby nedocházelo k nesrovnalostem, jsou datové pakety v síti očíslovány postupně, aby bylo možné určit pořadí zápisů. Existují ovšem některé problémy s nekonzistencí, na které systém může trpět:

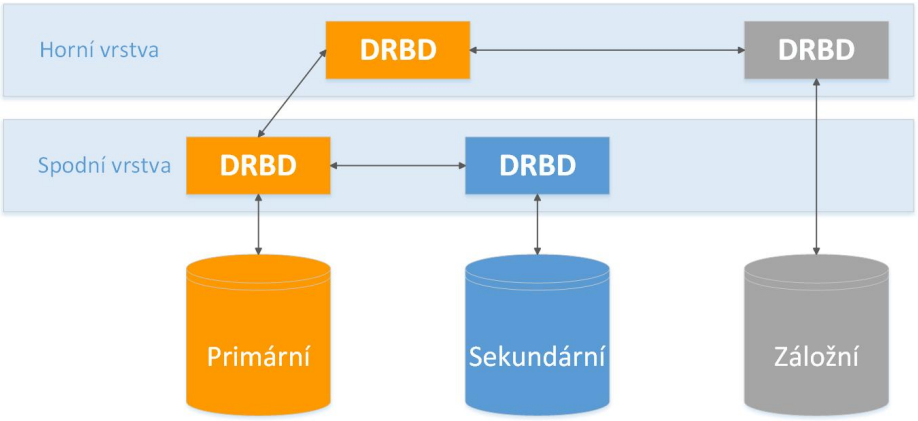
- **drbd_receiver:** Zpracovává příchozí pakety. Na sekundárním uzlu přiděluje velikost vyrovnávací paměti, přijímá bloky dat a vydává požadavky na zápis na místní disk.
- **drbd_sender:** Odpověď odesílatele pro datové bloky v odpovědi na požadavek čtení. To se provádí v jiném vláknu než drbd_receiver, aby nedošlo k distribuovanému zablokování. Pokud je spuštěn proces resynchronizace, pakety jsou generovány tímto podprocesem.
- **drbd_asender:** Potvrzení odesílatele. Ovladače pevného disku jsou informovány o dokončení požadavků prostřednictvím přerušení.

Pokud sekundární uzel umírá, neovlivní systém jako celek, protože zápisy nejsou iniciovány sekundárním uzlem. Jde-li o selhání primárního uzlu, data, která nedokončí zápis na disk, se mohou ztratit. Aby se tomu zabránilo, DRBD udržuje "protokol činností", vyhradí oblast na místním disku obsahující informace o operacích zápisu, které nebyly dokončeny. Každá změna způsobí aktualizaci metadat. DRBD synchronizuje data po obnovení havarovaného uzlu nebo v reakci na nesrovnalosti dat způsobené přerušením propojení. Rychlost synchronizace je konfigurována v konfiguračním souboru, viz [12].

Část komunikace mezi uzly je řešena vlákny, aby se zabránilo zablokování a složitému návrhu. Vlákna používaná pro komunikaci jsou:

- Simultánní zápis obou uzlů současně. V takovém případě je jeden z uzlů s právy zápisu odložen. Což způsobí zrušení požadavku zápisu při detekci současného zápisu a informuje, že zápis byl vyřazen.
- Lokální uzel zapíše daný blok a odesílá operaci zápisu do druhého uzlu. Vzdálený uzel potvrdí dokončení požadavku a pošle nový vlastní zápis do stejného bloku, to vše před dokončením místního zápisu. V tomto případě lokální uzel podrží požadavek na nový zápis dat, dokud nejsou místní zápisy dokončeny.

DRBD umožňuje také třífázovou replikaci. Při použití třífázové replikace přidává DRBD třetí uzel do existujícího clusteru se dvěma uzly a replikuje data do tohoto uzlu, kde jej lze použít pro účely zálohování a obnovení po havárii.



Obrázek 2.5: *Schéma DRBD pro třífázovou replikaci*

2.2.4 GlusterFS

Gluster je škálovatelný síťový souborový systém. Pomocí běžného externího hardwaru dokáže vytvářet rozsáhlé a distribuované řešení pro ukládání dat pro streamování médií, analýzu dat a další úlohy náročné na data a šířku pásma. GlusterFS propojuje úložné servery přes TCP/IP nebo RDMA (Remote direct memory access). Správa dat je centralizována v rámci celého clusteru, takže jsou přístupná v různých místech ukládání. Podporuje vícenásobná úložiště tím, že dělí uživatele a skupiny do logických svazků ve sdíleném úložišti, což eliminuje závislost uživatelů na tradičních paměťových polích. Kapacita může být přidána, aniž by byla ovlivněna výkonnost, čímž se zvýší účinnost a kapacita úložiště.

Soubory lze replikovat dvakrát nebo i vícekrát, aby bylo zajištěno, že data budou vždy k dispozici a to i v případě selhání hardwaru. Obnova je postupně prováděna na pozadí s téměř žádnou režií. GlusterFS nepoužívá proprietární formát pro ukládání souborů na disk, spíše používá v operačním systému podsystémový souborový systém např. ext3, zfs, takže data jsou vždy dostupná pomocí standardních nástrojů. Místo použití centralizovaného nebo distribuovaného serverů metadat GlusterFS používá algoritmus elastického hashovaní k vyhledání dat v úložišti. Data jsou uložena v logických svazcích, které jsou odvozeny od hardwaru

a jsou logicky odděleny od sebe. Serverová úložiště lze přidávat nebo odstraňovat, zatímco data zůstávají online bez přerušení aplikace. Rozhraní příkazového řádku (CLI), aplikační programovací rozhraní (API) a shell jsou sloučeny do jediného výkonného rozhraní umožňujícího automatizaci tím, že poskytuje CLI vyšší úrovně API a skriptovací schopnosti. Jazyky jako Python, Ruby nebo PHP lze použít k napsání série příkazů vyvolaných příkazovým řádkem.

GlusterFS pracuje s jednotkou zvanou *brick* (cihla), na jednom serveru může být takových to bricků více. Nachází se napříč všemi servery v clusteru a na základě konfigurace jsou spojeny do svazků, které jsou přístupné jako klasický souborový systém skrze protokol GlusterFS, nebo NFS. Bricky jsou tvořeny adresářem na lokálním souborovém systému. GlusterFS podporuje rozložení, která je možné zprovoznit i na jednom stroji, viz [12].

- **Distributed volume**: Pro každý soubor platí, že jeho data existují právě na jednom bricku. Využíváno pro velikou škálovatelnost.
- **Replicated volume:** Každý soubor se nachází na všech briccích. Využíváno je-li prioritou HA.
- **Stripped volume**: Narozdíl od Distributed volume může být jeden soubor uložen napříč bricky. Využíváno, je-li třeba ukládat soubory větší, než jsou velikosti disků na strojích.
- **Distributed Striped volume**: Kombinace, kde je kladen důrazem na výkon a obrovské soubory.
- **Distributed Replicated volume**: Kombinace, jedná se v podstatě o RAID 10.
- **Geo-replication**: Asynchronní jednosměrná replikace. Cílový stroj nemusí být připojen do clusteru. Poskytuje asynchronní replikaci dat přes geograficky odlišné lokality. Pracuje hlavně přes síť WAN a používá se k replikaci celého svazku, na rozdíl od automatické replikace souborů (AFR), která je replikací uvnitř clusteru. To je hlavně užitečné pro zálohování všech dat pro obnovu po havárii.

2.3 Ceph

Ceph je sjednocený distribuovaný úložný systém navržený pro vynikající výkon, spolehlivost a škálovatelnost. Ceph byl integrován s jádrem Linuxu, KVM a standardně zahrnut v mnoha distribucích GNU / Linux. Ceph je založen na RADOS (Reliable Autonomic Distributed Object Store), který poskytuje aplikacím úložiště objektů, bloků a souborových systémů v jediném unifikovaném úložišti clusteru, čímž je Ceph flexibilní, vysoce spolehlivý a snadno ovladatelný. Ceph RADOS poskytuje mimořádnou škálovatelnost ukládání dat – tisíce klientských hostitelů nebo KVM, kteří přistupují k PB až EB dat. Každá z aplikací může současně využívat rozhraní objektů, bloků nebo souborových systémů se stejným klastrem RADOS. Ceph automaticky replikuje data přes cluster. Také integruje s virtuálními zařízeními (KVM) na bázi jádra. Ceph poskytuje jednotný úložný systém, který zvládne ukládání souborů, bloků a objektů. Se zvyšující se kapacitou se nezvyšuje čas odezvy. Výkon společnosti netrpí tím, že datové úložiště roste, což z něj činí vhodný prvek pro projekty Big Data, viz [31].

Ceph používá standardní replikační model master/slave. Jeden server (uzel) funguje jako administrační server. Z tohoto serveru mohou být služby Ceph spravovány a nasazovány na více serverů (uzlů).

Ceph používá čtyři různé démony:

- **Cluster monitor**: Sleduje stav uzlů v clusteru.
- **Metadata sever**: Ukládá všechny metadata pro uzly a adresáře.
- **Object storage devices**: Uchovávají skutečná data.
- **RESTful gateways**: Spravují interakce mezi Ceph cluster a Amazon S3 nebo Swift-kompatibilní API.

Jednou z výhod úložiště Cephu je to, že dokáže zpracovat dohromady objektové úložiště i blokové úložiště. Většina distribuovaných škálovatelných úložných systémů může pracovat pouze s jedním nebo druhým. Tento unifikovaný úložný systém pomáhá eliminovat některé problémy při práci s Big Data.

Objektové úložiště se týká dat, která jsou uložena kdekoli na zařízení, přístupná prostřednictvím metadat a jedinečného identifikátoru. Tento způsob je obvykle rychlý a usnadňuje přístup a manipulaci s daty za běhu. Uložení dat jako objektů je také odolné, protože data jsou uložena v několika kopiích v systému. Pokud jeden uzel zmizí, nejsou ztraceny nebo nejsou k dispozici žádné údaje.

Blokové úložiště se odkazuje na data, která jsou uložena v určitém místě na daném fyzickém disku. Každý soubor je rozdělen na bloky a každý blok je uložen na své vlastní jedinečné adrese. Je to forma fyzické diskové jednotky stolního počítače. Blokové úložiště lze operačním systémem použít jako připojený diskový svazek.

Ceph souborový systém je podobný systému souborů NFS. To umožňuje hladce integrovat úložiště Ceph do libovolné aplikace, která používá příkazy POSIX k interakci se souborovým systémem. Tento souborový systém je spravován serverem metadat. Server metadat mapuje všechny adresáře a názvy souborů na objekty uložené v clusteru Ceph. Pro optimální běh celého clusteru je potřebné mít tři a více uzlů.

2.3.1 Proxmox

Proxmox Server Solutions je poskytovatel softwaru určený k vývoji výkonných a výkonných serverových řešení s otevřeným zdrojovým kódem. Proxmox VE (Virtual Environment) je kompletní open source platforma pro všestrannou virtualizaci, která integruje KVM a QEMU hypervisor a LXC kontejnery, dále softwarově definované úložiště a síťové funkce. To vše se nachází na jedné platformě, snadno spravuje clustery s vysokou dostupností a nástroje pro obnovu po havárii jsou integrované spolu s webovým rozhraním managementu. Kombinací dvou virtualizačních technologií pod jednou platformou nabízí maximální flexibilitu. Používá plnou virtualizaci KVM pro obrazy Windows a Linux a kontejnery pro spouštění aplikací Linux bez konfliktu. Systém je postaven nad jádrem operačního systému Debian. Proxmox VE je open source software a zdrojový kód je volně publikován pod GNU.

Proxmox VE používá přemostěný síťový model. Všechny VM (Virtual machine) mohou sdílet stejný most, jako kdyby byly virtuální kabely od každého hosta zapojeny do stejného přepínače. Most je pak připojen k fyzickým síťovým adaptérům hostitelského serveru, kterým je přiřazena konfigurace sítě (TCP/IP), takže VM mohou komunikovat s okolním světem.

Proxmox integruje nástroj pro zálohování nazvaný *utump*, který vytváří snímky virtuálních hostů jak pro Openvz, tak pro KVM. Nástroj utump vytvoří kopie dat VM, jež obsahují virtuální disky a všechna konfigurační data. Data jsou tak chráněna před ztrátou, viz [21].

2.3.2 **oVirt**

oVirt je stejně jako Proxmox kompletní platforma pro správu virtualizace, licencovaná a vyvinutá jako software s otevřeným zdrojovým kódem. oVirt se staví na virtuálním stroji založeném na jádru (KVM hypervisor) a na řídícím serveru RHEV-M, který Red Hat vydává komunitě s otevřeným zdrojovým kódem. oVirt Engine je řídicím centrem prostředí vVirt. Umožňuje definovat hostitele, konfigurovat datová centra, přidávat úložiště, definovat sítě, vytvářet virtuální počítače, spravovat oprávnění uživatelů a používat šablony z jednoho centrálního umístění. Ovirt je se systémem Proxmox takřka srovnatelný, hlavně co se týče funkcionality, ovšem má větší hardwarové nároky, viz [24].

3 VoIP

Zatímco tradiční telefonní služba komprimuje hlas do frekvence na drátě, VoIP komprimuje zvuk hlasu do paketů dat. Tyto komprimované datové pakety jsou pak odesílány přes internet. Když data dosáhnou konečného cíle, převedou zpět na zvuk. Pokud je používána služba VoIP k telefonování v tradiční telefonní síti (PSTN nebo veřejná přepínaná telefonní síť), volání VoIP se po dosažení sítě přepne na zvuk a hovor je normálně směrován. Rozdíl spočívá v tom, že uživatel zaplatil mnohem méně za toto volání pomocí poskytovatele služeb VoIP, viz [4].

Hlas/signál, který přichází přes VoIP, je digitalizován, aby byl směrován přes internet, nebo intranet. Existují dva standardy digitálního kódování, které se nejčastěji používají pro VoIP - H.323 a SIP (Session Initiation Protocol).

3.1 H.323

H.323 byl vyvinut před mnoha lety především pro videokonference přes ISDN telefonní linky. Vzhledem k tomu, že H.323 byla postavena už dávno, je její technologie považována za zastaralou a přichází s poměrně zbytečnými režijními náklady postrádajícími vlastnosti a flexibilitu, které SIP nabízí. Před protokolem H.323 byla všechna řešení VoIP založena na vlastních signalizačních protokolech a byla nekompatibilní. V roce 1996 se však s první verzí H.323 standard stal široce dostupným.

Standard H.323 je založen na čtyřech komponentech používaných ke spouštění hovorů a videokonferencí, např. konference typu point-to-point nebo multipoint:

- Endpoint je nástroj pro správu zařízení H.323 (jakési uživatelské rozhraní). Koncové body se mohou navzájem propojovat pomocí režimu VoIP – telefonování nebo videokonference.
- Gateways se používají k připojení koncových bodů z různých sítí, např. H.323 a ISDN.
- Zone controller nebo gatekeeper – je ústředním bodem sítě H.323. Je odpovědný za sestavení hovorů, správu šířky pásma a definování autentičnosti koncových bodů a bran v průběhu připojení. Přestože doporučení H.323 nestanoví bránu jako požadovaný prvek, nelze použít mnoho moderních funkcí implementovaných ve VoIP aplikacích a řešeních videokonferencí bez gatekeeperu.
- MCU (Multipoint Control Unit) – server se používá k připojení tří nebo více koncových bodů během jedné relace. Všechny koncové body účastnící se konference jsou nejprve připojeny k serveru MCU a MCU distribuuje video toky do všech koncových bodů. MCU provádí překódování video streamů, aby bylo možné mixovat a snižovat počet ostatních účastníků videa v jedinečném uspořádání pro každého účastníka.

Podle doporučení H.323 je mediální přenos dat rozdělen do pěti základních kroků:

- detekce a registrace brány,
- vytvoření spojení mezi dvěma nebo více koncovými body,
- výměna videa a hlasu prostřednictvím přepravních protokolů,
- multimediální výměna (přenos a spolupráce grafických a textových dokumentů,
- ukončení hovoru.

Funkce přenosu zvuku je považována za hlavní funkci standardu H.323. Každý koncový bod tedy podporuje jeden kodek z rodiny G.7xx.

3.2 SIP

SIP, protokol pro zahájení relace, je protokol IETF (Internet Engineering Task Force) pro VoIP a další textové a multimediální relace, jako jsou instant messaging, video, online hry a další služby. SIP je protokol end-to-end signalizace klient/server. Je to mnohem méně komplikované, pokud jde o implementaci a programování z hlediska vývojáře. Současně poskytuje všechny funkce potřebné pro škálování a spolehlivou hlasovou komunikaci. Se současnými bezpečnostními obavami má SIP také lepší řešení, pokud jde o poskytování prostředí front-end proxy a NAT (Network Address Translation).

Protokol SIP je textově založený a výrazně se podobá protokolu HTTP. Zprávy jsou založeny na textu a mechanismus požadavků a odpovědí usnadňuje odstraňování problémů. Skutečný přenos dat probíhá protokolem TCP (Transmission Control Protocol) nebo protokolem UDP (User Datagram Protocol) na vrstvě 5 modelu OSI. Zprávy SIP popisují identitu účastníků hovoru a způsob, jakým mohou účastníci komunikovat prostřednictvím sítě IP. SDP (Session Description Protocol) definuje typ mediálních kanálů, které budou vytvořeny pro relaci – deklaruje, jaké kodeky jsou k dispozici a mohou být použity pro vytvoření hovoru. Po dokončení této výměny zpráv o instalaci se médium nahradí za použití dalšího protokolu, obvykle RTP (Real-Time Transmission Protocol). RTP je standard pro internetový protokol, který určuje způsob, jakým mohou programy spravovat přenos multimediálních dat v reálném čase. VoIP je obvykle hlas, ale může být také video.

SIP nabízí všechny možnosti funkcí telefonní za pomocí IP sítě:

- volání nebo přenos médií,
- konference,
- podržení hovoru.

SIP protokol má pět částí pro sestavení a ukončení komunikace:

- lokalizace uživatele – určení koncového systému, který bude použit pro hovor,
- dostupnost uživatele – určení dostupnosti systému,
- možnosti uživatele – Určení médií a parametrů, které budou pro hovor použity,
- nastavení relace – Stanovení parametrů relace od obou stranách,
- řízení relace – Vyvolání relace včetně převodu, ukončení a úpravy parametrů,

SIP je postaven na modelu požadavek/odpověď, kde každá transakce sestává z požadavku, který vyvolává funkci.

Tabulka 3.1: *Popis metod SIP protokolu*

Metoda	**Popis**
ACK	Potvrzení zpráv pro INVITE.
BYE	Ukončení spojení.
CANCEL	Zrušení žádosti o spojeni.
INVITE	Žádost o vytvoření spojení.
RE-INVITE	Změna běžícího spojení.
OPTIONS	Možnosti serveru.
REGISTER	Registrace koncového zařízení.

Ve výchozím nastavení jsou zprávy SIP odeslány na portu 5060, pokud jsou nešifrované. Šifrované zprávy jsou odesílány přes port 5061. V rámci adresy SIP lze zadat jiný než standardní port a tuto výchozí hodnotu přepsat. SIP podporuje UDP i TCP spojení, ale může úspěšně pracovat na prakticky jakémkoli typu přenosu.

4 Asterisk

Asterisk je open source softwarová ústředna, vytvořená společností Digium, Inc. a neustále rostoucí základnou pro uživatele a vývojáře. Digium investuje jak do rozvoje zdrojového kódu Asterisku, tak i do nízkonákladového telefonního hardwaru, který spolupracuje se společností Asterisk. Ústředna Asterisk běží na platformách Linux a jiných platformách Unix. Asterisk běží na pozadí systému Linux nebo Unix FreeBSD nebo OpenBSD. Většina funkcí je dnes založena na systému Linux. Připojit k pobočce Asterisk PBX se lze s rozhraním příkazového řádku nebo jedním ze serverových grafických rozhraní. K dispozici je také rozhraní TCP / IP pro správu, která aplikace Asterisk používá. To dává příležitost sledovat server Asterisk v reálném čase, vidět připojení, schopnost vytvářet a ukončovat ho.

Asterisk podporuje mnoho protokolů pro hlasové služby přes IP. Jsou zahrnuty oba signalizační protokoly jako H.323 a SIP a protokoly pro přenos médií jako RTP. Mediální toky mohou být kódovány mnoha různými algoritmy, od alaw / ulaw (G.711) po GSM a ILBC.

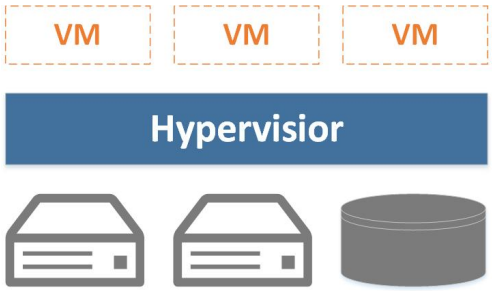
Asterisk může být aplikován jako:

- různorodá VoIP gateway (MGCP, SIP, IAX, H.323),
- pobočková ústředna (PBX),
- voicemail služby s adresářem,
- konferenční server,
- packet voice server,
- šifrování telefonních nebo faxových volání,
- voicemail služby s adresářem,
- interaktivní hlasový průvodce (IVR) server,
- softwarová ústředna (Softswitch),
- překlad čísel,
- aplikace Calling card,
- prediktivní volič (Predictive dialer),
- řazení volání do front se vzdáleným zprostředkovatelem.

5 Virtualizace

Virtualizace je technologie, která umožňuje vytvářet více simulovaných prostředí nebo vyčleněných zdrojů z jednoho fyzického hardwarového systému. Software nazvaný hypervisor se přímo připojuje k tomuto hardwaru a umožňuje rozdělit jeden systém na samostatná a bezpečná prostředí známá jako virtuální stroje. Tyto virtuální stroje se spoléhají na schopnost hypervisoru oddělit zdroje zařízení od hardwaru a vhodně je distribuovat.

Hypervisor je program pro vytváření a spouštění virtuálních strojů. Hypervisory jsou tradičně rozděleny do dvou tříd: hypervizor typu jedna nebo také *bare metal*, které spouštějí hostované virtuální stroje přímo na hardwaru systému a chovají se v podstatě jako operační systém. A typ dva nebo *hosted hypervisory*, ty se chovají spíše jako tradiční aplikace, které lze spustit a zastavit jako normální program. Když je virtuální prostředí spuštěno a uživatel nebo program vydá pokyny vyžadující další zdroje z fyzického prostředí, hypervisor přenese požadavek na fyzický systém a ukládá do paměti změny, které se dějí téměř nativní rychlostí nativního hardwaru, viz [27].

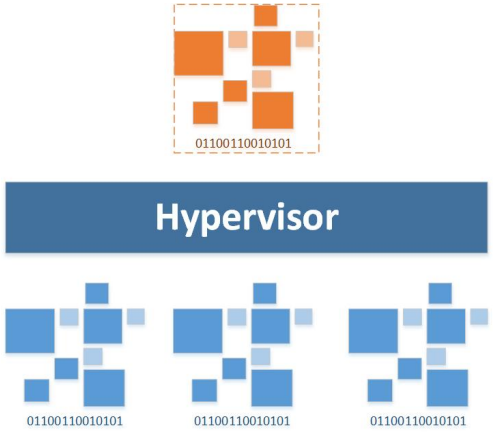


Obrázek 5.6: *Schéma popisující funkci virtualizace*

5.1 Typy virtualizace

5.1.1 Datová virtualizace

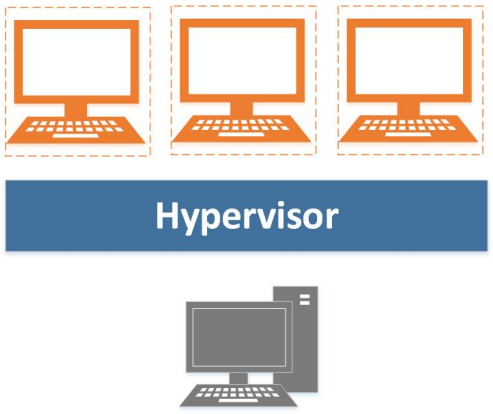
Data, která jsou rozšířena po celém světě, mohou být sloučena do jediného zdroje. Virtualizace dat umožňuje zpracovávat data jako veliké dynamické úložiště spojující data z více zdrojů do jednoho.



Obrázek 5.7: *Schéma datové virtualizace*

5.1.2 Virtualizace operačního systému

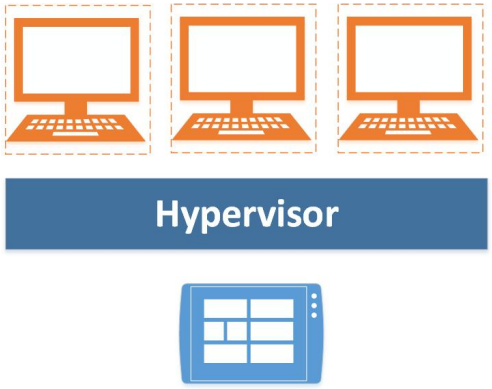
Virtualizace operačního systému se děje na jádře hostujícího operačního systému. Je to užitečný způsob, jak spustit prostředí Linux a Windows vedle sebe. Sníží se tak nároky na hardware. Zvyšuje se bezpečnost, protože všechny virtuální instance mohou být sledovány a izolovány.



Obrázek 5.8: *Schéma virtualizace operačního systému*

5.1.3 Desktop virtualizace

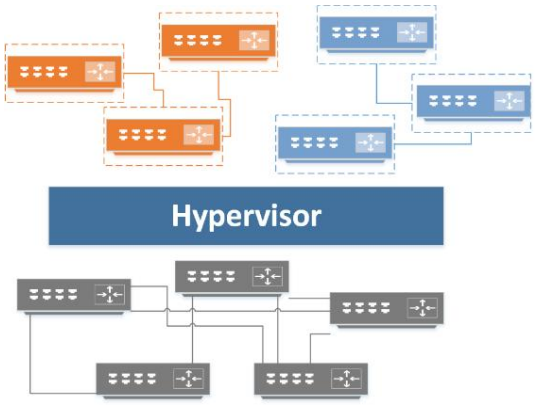
Snadno zaměňovaná s virtualizací operačního systému, která umožňuje nasazení více operačních systémů na jeden fyzický stroj. Virtualizace type desktop umožňuje centrálnímu administrátorovi (nebo automatizovanému administrativnímu nástroji) nasadit simulované desktopové prostředí na stovky fyzických strojů najednou. Na rozdíl od tradičních desktopových prostředí, která jsou fyzicky nainstalována, nakonfigurována a aktualizována na každém počítači, virtualizace typu desktop umožňuje správcům provádět hromadné konfigurace, aktualizace a kontroly zabezpečení všech virtuálních strojů.



Obrázek 5.9: *Schéma virtualizace typu desktop*

5.1.4 Virtualizace síťových funkcí

Funkce virtualizace síťových funkcí (NFV) odděluje klíčové funkce sítě, například adresářové služby, sdílení souborů a konfigurace IP. Jakmile jsou softwarové funkce nezávislé na fyzických počítačích, konkrétní funkce mohou být zabaleny společně do nové sítě a přiřazeny k virtuálním prostředí. Virtualizační sítě snižují počet fyzických komponent, jako jsou přepínače, směrovače, servery, kabely a rozbočovače potřebné k vytvoření více nezávislých sítí.



Obrázek 5.10: *Schéma virtualizace síťových funkcí*

5.2 KVM

KVM (Kernel-based Virtual Machine) je plně virtualizační řešení pro Linux na x86 architektuře hardwaru obsahujících rozšíření virtualizace (Intel VT nebo AMD-V). Skládá se z načteného modulu jádra, kvm.ko, který poskytuje základní virtualizační infrastrukturu a procesor specifický modul kvm-intel.ko, nebo kvm-amd.ko.

Pomocí KVM lze spustit více virtuálních strojů, které běží bez modifikovaných obrazů systému Linux nebo Windows. Každý virtuální počítač má soukromý virtualizovaný hardware: síťovou kartu, disk, grafický adaptér atd.

5.3 QEMU

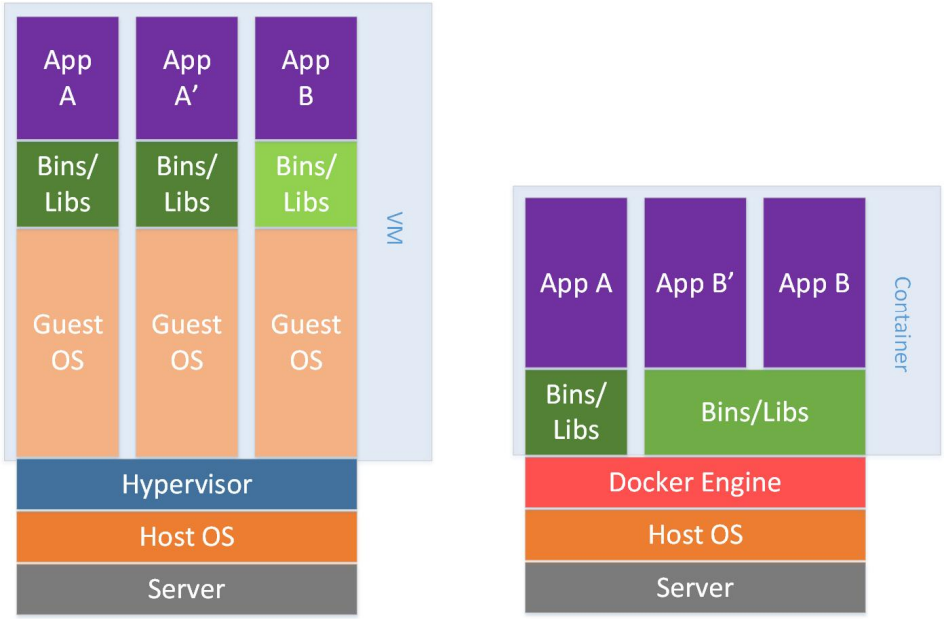
QEMU je generický a open source emulátor stroje a virtualizační nástroj. Při použití jako emulátoru stroje může QEMU spustit operační systémy a programy pro jeden počítač (např. Deska ARM) na jiném počítači. Pomocí dynamického překladu dosahuje velmi dobrého výkonu.

Při použití jako virtualizační nástroj dosahuje QEMU téměř nativní výkon spouštěním kódu hosta přímo na hostitelském CPU. QEMU podporuje virtualizaci při spouštění pod hypervisorem Xen, nebo pomocí modulu jádra KVM v Linuxu. Při použití KVM může QEMU virtualizovat x86, server a embedded PowerPC, 64bitové POWER, S390, 32bitové a 64bitové ARM a hosty MIPS.

6 Kontejner

Jedná se zjednodušeně řečeno o virtualizaci, u které se ovšem nesimuluje veškerý hardware, nad nímž potom běží celý virtuální počítač. Nabízí možnost, jak izolovat software od hostitelského operačního systému, na kterém jsou spuštěny. Obraz kontejneru je samostatný spustitelný balík softwaru obsahující vše potřebné ke spuštění, tzn. kód, databáze, systémové nástroje, systémové knihovny, nastavení. Jsou k dispozici pro aplikace založené na Linuxu a Windows. Kontejnery izolují software od svého okolí, například rozdíly mezi vývojovými a produkčním prostředími. Pomáhají snižovat konflikty mezi softwarem na stejné infrastruktuře a nemusí na rozdíl od virtualizace replikovat celý operační systém, ale pouze jednotlivé komponenty, které potřebují k provozu. Kontejner sdílí jádro operačního systému s ostatními kontejnery, sdílené části operačního systému jsou pouze pro čtení, což zabraňuje poškození hostujícího operačního systému. A také, že kontejnery jsou méně hardwarově náročné než virtualizace. Kontejner může mít pouze desítky megabajtů, zatímco virtuální počítač s vlastním celým operačním systémem může mít velikost několika gigabajtů. Z tohoto důvodu může jediný server hostit mnohem více kontejnerů než virtuální stroje. Dalším velkým přínosem je, že virtuálnímu stroji může trvat několik minut, než spustí své operační systémy a aplikace, které jsou hostitelem, zatímco kontejnerové aplikace mohou být spuštěny téměř okamžitě.

Technologie kontejneru byla vyvinuta již v roce 2001 jako projekt pro operační systém Linux. Veliký rozvoj přišel až s projektem zvaným Docker, který byl vypuštěn jako Open Source v roce 2013. Docker dokázal během velmi krátké doby nalákat velké množství vývojářů, včetně velkých hráčů, jako je Red Hat, nebo Amazon a tak se stal jakýmsi standardem. Úzce s Dockerem spolupracovala také firma Microsoft a technologii kontejneru implementovala do svého operačního systému Windows Server 2016, viz [29].



Obrázek 6.11: *Zobrazení rozdílů mezi VM a kontejnery*

6.1 Docker

Docker je nejznámější open source řešení pro správu kontejnerů. Poskytuje integrovanou, testovanou a certifikovanou platformu pro aplikace běžící na systémech Linux nebo Windows a poskytovatelích cloudových řešení. Docker je vydávám ve dvou verzích:

- Docker CE (Community Edition) je zdarma a ideální pro vývojáře a malé týmy, které chtějí začít s aplikací Docker a experimentují s aplikacemi založenými na kontejnerech. K dispozici je pro mnoho populárních infrastrukturních platforem, jako jsou desktopové, cloudové a open source operační systémy. Docker CE poskytuje instalaci pro jednoduchou a rychlou instalaci, takže dovoluje začít okamžitě rozvíjet.
- Docker EE (Enterprice Edition) je předplatné softwaru, podpory a certifikace pro podnikové vývojové a IT týmy vytvářející a spravující kritické aplikace ve výrobním měřítku. Docker EE je platforma typu Containers-as-a-Service pro IT, která spravuje a zabezpečuje různorodé aplikace napříč různorodou infrastrukturou. Nabízí intuitivní a snadno použitelné webové uživatelské rozhraní, zabezpečení typu end-to-end a mnoho dalších.

Docker má podporu pro operační systémy Linux, i Windows. Pro Docker s linuxovými kontejnery se většinou používá prosté označení *Docker*, popřípadě *Linux Containers*, pro nativní Docker na Windows pak *Windows Containers*. Pojem *Docker for Windows* se používá spíše pro nástroj umožňující provozovat linuxové kontejnery na Windows s pomocí virtualizačních nástrojů, které jsou implementovány ve verzi pro systém Windows.

Tyto kontejnery jsou vzájemně nekompatibilní, není možné provozovat linuxové kontejnery jako nativní procesy ve Windows a není možné provozovat Windows kontejnery na Linuxu. Nicméně je možné pohodlně provozovat linuxové kontejnery na Windows díky zmíněné virtualizaci. Virtuální stroj s Linuxem je spuštěn uvnitř Windows a v Linuxu je nainstalovaný Docker, s kterým si povídá Docker Tooling z nativního prostředí, tedy příkazové řádky Windows, viz [28].

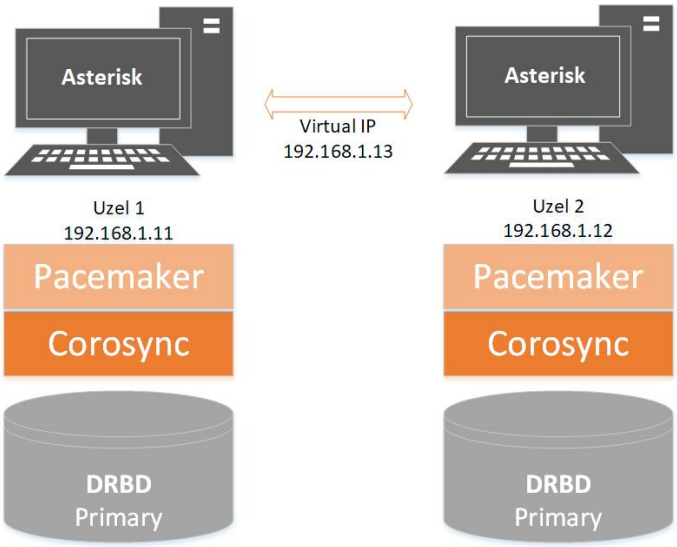
7 Realizace řešení vysoké dostupnosti

Pro otestování vysoké dostupnosti telefonní ústřednu Asterisk jsem zvolil celkem tři implementace, které jsou v tomto dokumentu popsány. Využil jsem odlišné technologie, aby byly otestovány klady a zápory aktuálních řešení. Ve všech implementacích jsou použita síťová úložiště s funkcí replikace. V každé z implementací jsem využil odlišný systém pro úložiště, abych je mohl otestovat, jelikož každé z úložišť má drobné změny ve funkčnosti a tak i jiné uplatnění. V ústředně Asterisk byly pozměněny pouze konfigurační soubory, to u jednotlivých implementací s minimálními rozdíly. Byla použita výchozí databáze SQLite, abych ověřil její vliv v jednotlivých řešení.

Testovaná řešení pro vysokou dostupnost ústředny byly výhradně realizována na virtualizovaném prostředí open source softwaru VirtualBox ve verzi 5.2 s použitím virtualizace typu KVM. Každému virtuálnímu stroji byl přiřazen jeden procesor, 1024 MB operační paměti a dva pevné disky o velikosti 15 GB. Jako VoIP ústřednu jsem zvolil Asterisk ve verzi 13.2 LTS, z důvodu dlouhodobé podpory. Ústředny byly konfigurovány dle potřeb jednotlivých implementací. Jako výchozí operační systém pro ústředny byl použit linuxový systém Ubuntu ve verzi 16.10 se šedesáti čtyř bitovou architekturou.

7.1 Implementace DRBD a Pacemakeru

Toto řešení je založeno na replikaci dat s použitím sdílené virtuální IP adresy. Dále technologie Pacemaker a Corosync pro sledování dostupnosti ústředen, kde Corosync obstarává komunikaci všech uzlů mezi sebou v rámci clusteru. Pacemaker provádí určité úkony, jako například přesun virtuální IP adresy, změna role, či spuštění skriptu a jiných softwaru. Pokud tedy nastane předem definovaná událost, Corosync jí oznámí ostatním uzlům, Pacemaker úkon zjistí a provede.



Obrázek 7.12: *Schéma implementace DRBD a Pacemakeru*

V tomto případě byly vytvořeny dva virtuální servery s operačním systémem Ubuntu, v roli master/slave. V případě výpadku hlavního serveru se automaticky převede za pomocí pacemakeru role master na záložní server, také se přesune virtuální IP adresa. Všechny potřebné konfigurační soubory ústředny Asterisk jsou uloženy v replikovaném úložišti. Obě ústředny tak využívají stejná nastavení.

Tabulka 7.2: *Přehled síťové adresace*

Hostname	Popis	IP adresa	Maska	Brána
Ubuntu1	Virtuální stroj č.1	192.168.1.11	255.255.255.0	192.168.1.1
Ubuntu2	Virtuální stroj č.2	192.168.1.12	255.255.255.0	192.168.1.1
-	Virtuální IP adresa	192.168.1.13	255.255.255.0	192.168.1.1

7.1.1 Konfigurace Pacemakeru a Corosync

Kdykoli je použito více serverů komunikujících mezi sebou, obzvláště s clusterovým softwarem, je důležité zajistit, aby byl synchronizován čas na všech stanicích. Pro synchronizaci našich serverů použijeme NTP (Network Time Protocol).

```
$ sudo apt-get -y install ntp
```

Corosync využívá přenos UDP mezi porty 5404, 5405 a 5406. Pokud je konfigurována brána firewall, je nutné se ujistit, že komunikace na těchto portech je povolená.

```
$ sudo iptables -A INPUT  -i eth1 -p udp -m multiport --dports
5404,5405,5406 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

```
$ sudo iptables -A OUTPUT  -o eth1 -p udp -m multiport --sports
5404,5405,5406 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

Corosync a Pacemaker jsou na sobě závislé, takže se instalují pomocí jednoho instalačního balíčku.

```
$ sudo apt-get install pacemaker
```

Corosync musí být nakonfigurován tak, aby servery mohly komunikovat jako cluster. Proto je nutné upravit konfigurační soubor */etc/hosts*, který obsahuje názvy uzlů a jejich IP adresy. Poté je vygenerován autorizační klíč clusteru o velikosti 128b a zapsán do adresáře */etc/corosync/authkey*. Klíč musí být vygenerován na hlavním uzlu a poté překopírován na zbylý záložní uzel do zmíněného adresáře. Adresáři je nutné přidělit příslušná práva.

Na obou uzlech provedeme úpravu souboru */etc/corosync/corosync.conf*, která bude na všech uzlech stejná. Je nutné definovat, název clusteru, virtuální IP adresu a privátní IP adresu obou uzlů, zbylým hodnotám můžeme ponechat výchozí hodnotu. Kompletní konfigurační soubor je přiložen jako příloha. [Příloha A]

```
cluster_name: mycluster                    /název clusteru
bindnetaddr: 192.168.1.13                  /virtuální IP adresa
ring0_addr: 192.168.1.11                /privátní IP adresa uzlu
name: primary                           /název uzlu
nodeid: 1                               /identifikační číslo uzlu
```

Dále je třeba nakonfigurovat Corosync, aby se spojil se službou Pacemaker. Na obou uzlech clusteru vytvoříme soubor pcmk v adresáři služeb Corosync a vložíme konfiguraci. Konfigurační soubor je přiložen v příloze.

```
$ sudo mkdir -p /etc/corosync/service.d
$ sudo vim /etc/corosync/service.d/pcmk
```

Nakonec otevřeme soubor */etc/default/corosync*, přidáme řádek *START=yes*, který povolí zpuštění a po uložení změn Corosync spustíme. Pokud řádek existuje, postačí jej upravit.

```
$ sudo service corosync start
```

Nastavíme, aby se služby spouštěly automaticky po startu serveru. Vzhledem k tomu, že Pacemaker musí být spuštěn po Corosync, nastavíme prioritu startu Pacemakerova na 20, což je vyšší než priorita Corosync (výchozí nastavená hodnota je 19).

```
$ sudo update-rc.d pacemaker defaults 20 01
```

Musíme také nakonfigurovat některé výchozí nastavení, ještě před prvním spuštěním, jelikož by pacemaker zahlásil chybu. Funkce *stonith-enabled* je ve výchozím stavu povolena, ovšem k jejímu použití je nutná další konfigurace, kterou k tomuto testování není potřeba. Stonith (Shoot The Other Node In The Head) neboli fencing, což je soubor metod, které chrání data před neúmyslným souběžným přístupem, viz [35]. Tomuto problému dále zabraňuje samotné síťové úložiště. Tento cluster se skládá ze dvou uzlů, proto musí mít nastaveno, aby ignorovaly politiky pro quorum, jinak by totiž při odstavení jednoho z nodů zastavil službu i na druhém nodu.

```
$ sudo crm configure property stonith-enabled=false
$ sudo crm configure property no-quorum-policy=ignore
```

Chcete-li komunikovat s Pacemaker, použijeme nástroj CRM. Nástroj spustíme pouze na jednom ze serveru a nastavíme virtuální IP adresu, která je nastavena vždy na serveru v roli master. Pokud tedy hlavní server selže, virtuální IP adresu převezme záložní server. V této ukázce vidíme zvolenou virtuální IP adresu, dále masku sítě, do které adresa patří. Parametr monitor interval nám říká, v jaké časové úseku bude virtuální adresa monitorována, čas jen měřen od posledního selhání. Parametr se ovšem vztahuje jen k IP adrese, nikoliv ke stavu uzlu. Časový parametr *failure-timeout* udává čas v sekundách, po který vyčkává, než zdroj opět povolí a začne se chovat, jako by se opět nic nestalo. A *resource-stickerss* kontroluje, zda se musí zdroj vrátit zpět na uzel, kde byl dříve spuštěn po obnovení, nebo ne. V tomto případě se zdroj zpět nevrací.

```
$ sudo crm configure primitive virtual_public_ip \
ocf:heartbeat:IPaddr2 params ip="192.168.1.13" \
cidr_netmask="24" op monitor interval="2s" \
meta migration-threshold="2" failure-timeout="30s" resource-
stickiness="100"
```

Pomocí příkazu *sudo crm status* otestujeme stav serveru.

Z výpisu nastavení IP adresy vidíme, že server v roli master je držitelem virtuální IP adresy, pomocí které je teď přístupný v síti. Pokud je vše správně nastaveno, zobrazí stav zdroje.

```
Online: [ primary secondary ]
Full list of resources:
virtual_public_ip     (ocf::heartbeat:IPaddr2):  Started primary
```

Výpis oznamuje, že služba je aktivní, oba uzly jsou v pořádku a virtuální adresa je nastavena na primárním uzlu. Pokud si tedy zobrazíme stavení IP adres, zjistíme, že je virtuální IP adresa jen nastavena. Pro kontrolu se můžeme na ni dotázat z kteréhokoliv počítače v síti.

```
lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
inet 127.0.0.1/8 scope host lo valid_lft forever preferred_lft
forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP group default qlen 1000
    inet 192.168.1.12/24 brd 192.168.1.255 scope global enp0s3
    valid_lft forever preferred_lft forever
    inet 192.168.1.13/24 brd 192.168.1.255 scope global
secondary
    enp0s3 valid_lft forever
```

Provedl jsem také kontrolu pomocí síťového analyzátoru Wireshark. Z přiloženého výpisu komunikace mezi uzly clustery vidíme dotazy o stavu protějšího uzlu. Pokud není obdržena odpověď, dotaz je opakován vícekrát. Poté vyhodnotí nečinnost. Převedení virtuální IP adresy trvá v řádu milisekund, až sekund.

NO.	*Time*	*Source*	*Destination*	*Protocol*	*Lenght*	*Info*
87	*5.609954808*	*192.168.1.11*	*192.168.1.12*	*UDP*	*162*	*5405 → 5405*
88	*5.610145490*	*192.168.1.12*	*192.168.1.11*	*UDP*	*162*	*5405 → 5405*
89	*5.848309597*	*192.168.1.12*	*192.168.1.11*	*UDP*	*162*	*5405 → 5405*
90	*5.848309597*	*192.168.1.12*	*192.168.1.11*	*UDP*	*162*	*5405 → 5405*

7.1.2 DRBD

Pro replikaci dat jsem v tomto případě využil systém DRBD, který je k tomu určený. Tento systém je lépe přizpůsoben pro použití v clusteru o dvou uzlech. V případě výpadku jednoho z uzlů jsou data stále zálohována a dle nastavení přístupná jinému uzlu.

Jak bylo zmíněno v kapitole 2.2.3 systém DRBD umí pracovat ve dvou režimech: Single-primary a Dual-primary mód. Systém DRBD jsem poprvé testoval v režimu single-primary spolu se systémem Pacemaker. Při výpadku serveru byla primární role převedena na záložní uzel a ten mohl přistupovat k datům. Avšak objevil se problém u ústředny Asterisk, která využívá databázi postgresql. Výhodou této databáze je její jednoduchost. Všechny záznamy jsou uloženy v jediném souboru, který lze přesunout na replikované úložiště. Ovšem zde nastává problém. Pokud spustíme Asterisk na obou členech clusteru s tím, že jeden člen nemá možnost zápisu, tak Asterisk nelze spustit a zahlásí chybu. Z tohoto důvodu jsem poté zvolil režim dual-primary se souborovým systémem OCFS2, ve kterém mají oba členové možnost zápisum.

Nejprve musíme nainstalovat a nastavit DRBD se dvěma primárními uzly a OCFS2 jako náš souborový systém. OCFS2 je sdílený diskový souborový systém vyvinutý společností Oracle Corporation a uvolněný pod GNU. Tento souborový systém jsem zvolil z důvodu snazší implementace. Začneme tím, že vytvoříme soubor *datastore.res* a provedeme konfiguraci. Soubor uložíme do adresáře */etc/drbd.d/*.

V konfiguračním souboru je nutné upravit název virtuálního disku (můžeme ponechat výchozí), název disku, kde budou data ukládána. V tomto případě byl využit druhý pevný disk *sdb1*, který byl přidělen virtuálnímu stroji. Taktéž je potřeba vložit IP adresy obou uzlů. Celý konfigurační soubor je přiložen v příloze. [Příloha B]

```
Logical common/shared device for storage
        device  /dev/drbd0;
        disk /dev/sdb1;
        allow-two-primaries yes;

        on node01 { address 192.168.1.11:7789; }
        on node02 { address 192.168.1.12:7789; }
```

Nedoporučuji nastavit volbu *allow-two-primaries* na hodnotu *yes* při počáteční konfiguraci. Volba se povolí až po dokončení synchronizace. Mohou nastat komplikace a počáteční synchronizace proběhne rychleji.

Poté provedeme úpravu souboru */etc/drbd.d/global_common.conf*. Ten je nakonfigurován tak, aby synchronizace byla, co nejbezpečnější pomocí protokolu *C*. Synchronizace je pomalejší, ale bezpečnější. Protokol je popsán v kapitole 2.2.3.

Nastavíme diskový oddíl, který DRBD používá pro replikaci v obou uzlech a spustíme počáteční synchronizaci. Ta trvá dle velikosti oddílu. Po dokončení synchronizace upravíme na obou uzlech soubor datastore.res a povolíme parametry, které umožňují primární ukládání.

```
$ e2fsck -f /dev/sdb1
$ resizefs /dev/sdb1 15GB
$ /etc/init.d/drbd start
$ drbdsetup /dev/dbrd0 primary –o
```

Ubuntu ve verzi 14 a 16 nepodporuje správce zámku dat pro Pacemaker, takže použijeme starší možnost OCFS2 DLM. OCFS2 využívá centrální konfigurační soubor, */etc/ocfs2/cluster.conf*. V souboru opět změníme IP adresy a názvy uzlů. Konfigurační soubor je přiložen v příloze.

```
node:
    ip_port = 7777
    ip_address = 192.168.1.11
    number = 0
    name = Ubuntu1
    cluster = ocfs2
```

Teď je nutné službu DRBD spustit a vytvořený diskový oddíl se souborovým systémem OCFS2 připojit k námi vytvořené složce. Aby bylo možné se připojit do clusteru je nutné ještě restartovat službu o2cb.

```
$ /etc/init.d/o2cb restart
$ /etc/init.d/drbd start
$ mount.ocfs2 /dev/drbd0  /home/user/Asterisk
```

V tomto okamžiku je vše hotovo. Pro otestování OCFS2, vložíme jakýkoliv soubor do úložiště v jednom uzlu a zkontrolujte, zda se objeví na druhém uzlu. Pokud se soubor odebere, měl by být okamžitě odebrán na obou uzlech. DRBD je ve výchozím stavu nastaven, že se disk automaticky nenamapuje. Proto je nutné po restartu uzlu disk znovu namapovat, nebo nastavit automatické namapování po startu operačního systému.

7.1.3 Instalace a konfigurace ústředny Asterisk

Telefonní ústřednu Asterisk lze nainstalovat více způsoby. Z oficiálních zdrojů, kde stáhneme instalační soubory a poté nainstalujeme, tyto možnosti nabízí aktuální verze Asterisku 15. Nebo instalace z repositářů operačního systému, kde se nachází verze 13 s dlouhodobou podporou. V tomto případě zvolený způsob nehraje až tak velikou roli. Důležité však je nastavení ústředny.

Globální nastavení konfiguračních souborů Asterisku a jejich umístění se nachází v konfiguračním souboru */etc/asterisk/asterisk.conf*. Důležité hodnoty jsou vysvětleny

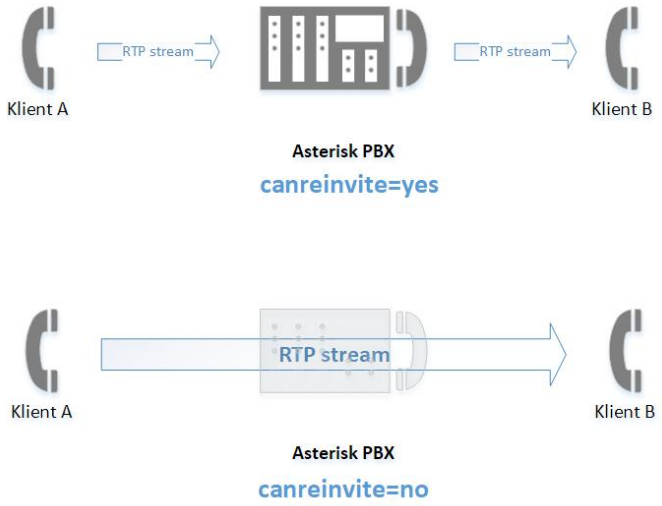
v přiložené tabulce. Soubory Asterisku je nutné přesunout na síťové úložiště, abychom zaručili, že nastavení budou u obou uzlů stejná. Každou ze složek můžeme připojit k úložišti, nebo mnou zvolený způsob, složky přesunout na síťové úložiště a cesty pozměnit v souboru *asterisk.conf*. Upravený konfigurační soubor je vložen v příloze [Příloha F]. Poslední tři hodnoty v tabulce není nutné upravovat, abychom mohli přečíst logovací soubory daného uzlu a bylo možné ústřednu spustit. V souboru je nutné odstranit znak *„(!)“*, který se nachází na začátku, jinak Asterisk nepřevezme upravené cesty k souborům.

Tabulka 7.3: *Konfigurační hodnoty Asterisku*

Hodnota	**Vysvětlení**
astetcdir	Umístění konfiguračních soubory Asterisku
astmoddir	Umístění použitých modulů
astvarlibdir	Umístění stavových informací
astdbdir	Umístění vnitřní databáze
astkeydir	Umístění šifrovacích klíčů
astdatadir	Umístění pomocných dat
astagidir	Umístění AGI skript
astspooldir	Umístění souborů pro hovory a záznamy
astrundir	Umístění čísla procesu
astlogdir	Umístění log souborů
astsbindir	Umístění systémových binárních souborů, které Asterisk využívá

Jedním z hlavních souborů pro úpravu je *SIP.conf*, ve kterém se nachází údaje o nastavení SIP a všech uživatelích. Abychom mohli uskutečnit hovor, musíme přidat klienty do tohoto souboru. Soubor je rozdělen na sekce. V sekci General jsou uloženy konfigurace týkající se celé ústředny Asterisk. Další sekce slouží k nastavení jednotlivých klientů. Více o konfiguraci naleznete v příloze. Důležitým aspektem v sekci General je řádek *canreinvite=no*.

Tato volba v souboru *sip.conf* se používá pro informování serveru Asterisk, že klienti nevyžadují zprávu reinvite po ukončení hovoru, pokud to není skutečně nutné. Navíc RTP stream neprochází přes ústřednu Asterik, což má za následek menší zátěž procesoru ústředny. A hlavně, pokud ústředna bude mít výpadek, probíhající hovory se neukončí, jelikož pobíhají jen mezí klienty. Ústředna slouží jen k sestavení. Nastává zde ovšem problém. Pokud dojde k výpadku ústředny, tak hovory přetrvají, ovšem když jeden z účastníků hovor ukončí, druhý účastník se o ukončení nedozví a musí tak hovor manuálně ukončit sám. Ve srovnání s výpadkem hovoru můžeme toto považovat za více vyhovující.



Obrázek 7.13: *Význam parametru canreinvite*

Dále jsem do souboru vložil tři klienty 1000, 2000 a Sipp.

```
[1000]
type=friend           /Definice účtu
language=cz           /Definice jazyka
host=dynamic          /Povolení registrace na všech adresách
secret=1000           /Zvolené heslo pro registraci
context=internal /Definice skupiny v dialplanu (extensions.conf)
```

Posledním souborem, který jsem upravoval, je soubor *extensions.conf*. Zde jsou konfigurovány takzvané dialplany udávající ústředně postup pro hovory. Definoval jsem zde dva dialplany. Dialplan internal sloužící k sestavení hovoru. Je potřebný také k otestování mezi dvěma softwarovými klienty, mezi nimiž byl hovor testován. A dialplan sipp, který hovor vyzvedne, provede definované úkony a hovor ukončí. Oba tyto soubory jsou vloženy v příloze. [Příloha D, E] Podrobnosti k testování jsou popsány v další kapitole.

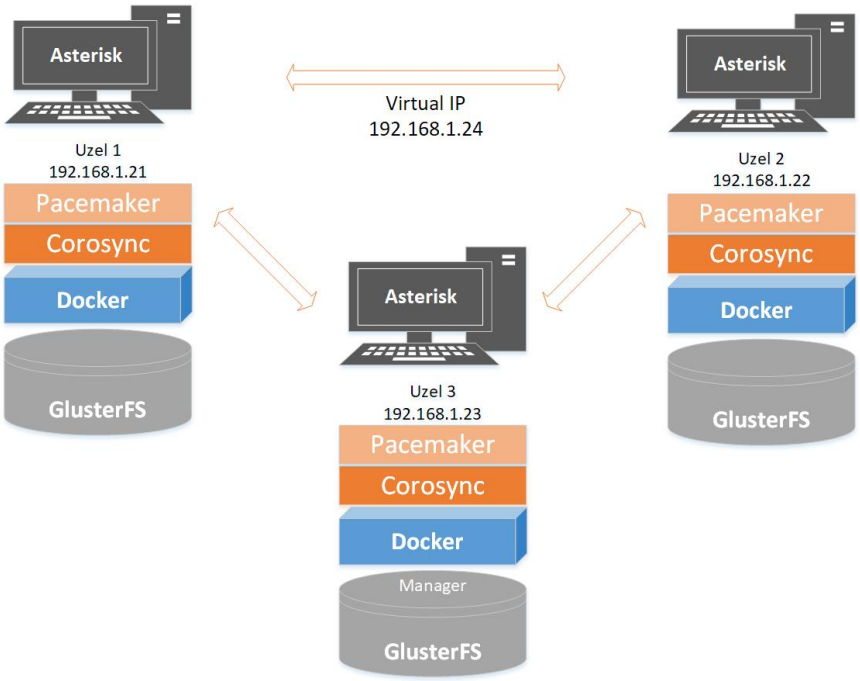
7.2 Implementace Docker

Jak již název napovídá, řešení je založeno na technologii kontejnerů a systému Docker. Tato technologie byla popsána v předchozí kapitole 6.1, nyní se tedy budeme věnovat její implementaci. Systém Docker nabízí vlastní řešení pro vysokou dostupnost a vyvážení zátěže nazývané swarm mode. To umožňuje spojení počítačů do clusteru a poté s nimi pracovat jako s jediným. Dříve existoval takzvaný Docker Swarm, což byl samostatný produkt, který jste mohli použít pro seskupování více počítačů s Dockerem. Seskupení bylo ovšem velmi složité a bylo potřeba mnoho nastavení. Proto Docker tuto technologii upravil, přidal mnoho vylepšení, jako například zabezpečení pomocí TLS a zabudování do nativní instalace od verze 1.12. Chcete-li spustit tento režim, stačí Docker naistalovat na více strojích, spustit ho v režimu pro cluster a jednotlivé stroje propojit. Samotné konfigurace, tvorbu a výměnu certifikátů si režim obstará sám.

Swarm mode má ovšem stále své nedokonalosti, které je nutné kompenzovat jinými nástroji. Ovšem Docker se stále vyvíjí a s rostoucí popularitou se tyto nedokonalosti odstraňují.

Jeden z prvních problému je, že Docker nedokáže pevně přiřadit IP adresu kontejneru a poté ho i s adresou přesunout. Využil jsem tedy opět systému Pacemaker a Corosync. Implementace probíhala stejně jako u předešlého řešení, kde je také popsána, kapitola 6.1.1.

Dalším kritickým místem je sdílení dat, jelikož pokud dojde k přesunu kontejneru, bude přesunuta jen základní image, nikoliv změny, které byly provedeny. K vyřešení jsem opět zvolil síťové úložiště s možností replikací dat. V tomto případě jsem použil systém glusterFS, který je lépe přizpůsoben pro práci s více jak dvěma uzly. Také jsem chtěl otestovat rozdíly v implementaci mezi jednotlivými síťovými úložišti.



Obrázek 7.14: *Schéma implementace Docker*

Implementace byla testována na třech virtuálních stojích, s následným nastavením:

Tabulka 7.4: *Přehled síťové adresace pro implementaci Docker*

Hostname	Popis	IP adresa	Maska	Brána
DockerUbuntu1	Virtuální stroj č.1	192.168.1.21	255.255.255.0	192.168.1.1
DockerUbuntu2	Virtuální stroj č.2	192.168.1.22	255.255.255.0	192.168.1.1
DockerUbuntu3	Virtuální stroj č.2	192.168.1.23	255.255.255.0	192.168.1.1
-	Virtuální IP adresa	192.168.1.24	255.255.255.0	192.168.1.1

7.2.1 GlusterFS

Pro instalace a správnou funkci je opět nutné upravit soubor */etc/hosts*, jenž je vložen v příloze [Příloha G]. Poté provedeme instalaci balíčku *glusterfs-server*, který je obsažen v repositářích. Po úspěšném dokončení instalce glusterFS spustíme. Vytvoříme také úložiště, kde se budou data ukládat, viz [20].

```
$ sudo apt install -y glusterfs-server
```

```
$ /etc/init.d/glusterfs-server start
```

```
$ sudo mkdir -p /gluster/data /swarm/volumes
```

Připravíme souborový systém pro úložiště GlusterFS na všech uzlech. Využijeme opět druhého pevného disku, který je přidám ke každému virtuálnímu stroji.

```
$ sudo mkfs.xfs /dev/sdb
```

```
$ sudo mount /dev/sdb /gluster/data/
```

A propojíme uzly do clusteru. Toto provedeme na prvním uzlu.

```
$ sudo gluster peer probe node2
```

```
peer probe: success.
```

Nyní vytvoříme jednotku a spustíme ji.

```
$ sudo gluster volume create swarm-vols replica 3
DockerUbuntu1:/gluster/data DockerUbuntu2:/gluster/data
DockerUbuntu3:/gluster/data force
```

```
$ sudo gluster volume start swarm-vols
```

Na každém z uzlu je nutné jednotku propojit s námi zvolenou složkou.

```
$ sudo mount.glusterfs localhost:/swarm-vols /swarm/volumes
```

Po připojení otestujeme funkčnost tím, že nakopírujeme libovolný soubor do zvolené složky a zkontrolujeme, zdali se propsal do všech uzlů v clusteru.

7.2.2 Docker Swarm mode

Pro implementaci vysoké dostupnosti je zapotřebí nejméně tří uzlů. Docker totiž využívá Raft Consensus algoritmus. Ten je navržen tak, aby byl snadno srozumitelný. Je to ekvivalent v odolnosti proti poruchám a výkonu. Rozdíl spočívá v tom, že je rozložen do relativně nezávislých podproblémů a čistě řeší všechny důležité součásti potřebné pro praktické systémy. Pro Docker je důležité, aby zajistil, že všechny uzly, které jsou odpovědné za správu a plánování úkolů v clusteru, uchovávají stejný konzistentní stav. Toleruje až *(N-1) / 2* selhání a vyžaduje většinu *(N / 2) + 1* členů, aby se dohodli na hodnotách navržených pro cluster. Příklady jsou uvedeny v tabulce 1.5.

Tabulka 7.5: *Přehled tolerovaných selhání*

Počet uzlů	Tolerovaný počet selhání
1	0
3	1
5	2
7	3

Docker lze opět nainstalovat dvěma způsoby, oficiální instalací, či pomocí depositářů. Tato ovšem nabízí zastaralou verzi, proto jsem zde použil možnosti instalace z oficiálních zdrojů. Instalaci jsem provedl na všech třech uzlech.

Po dokončení provedeme inicializaci docker swarm mode. To provedeme na jednom z uzlů, který si označíme jako manager. Role manager lze poté předat jinému uzlu.

```
$ docker swarm init --advertise-addr 192.168.1.20
```

V případě, že inicializace proběhla v pořádku, zobrazí se vygenerovaný řetězec, který je potřebný pro přidání zbylých uzlů do clusteru.

```
$ docker swarm join \
    --token SWMTKN-1-
49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-
8vxv8rssmk743ojnwacrr2e7c \
```

Jednotlivé uzly se tak spojí a provede se výměna klíčů pro zabezpečené spojení. Pro otestování můžeme provést výpis pomocí příkazu *$ docker node ls*. Ten vypíše všechny uzly, clustery, jejich stav a roli.

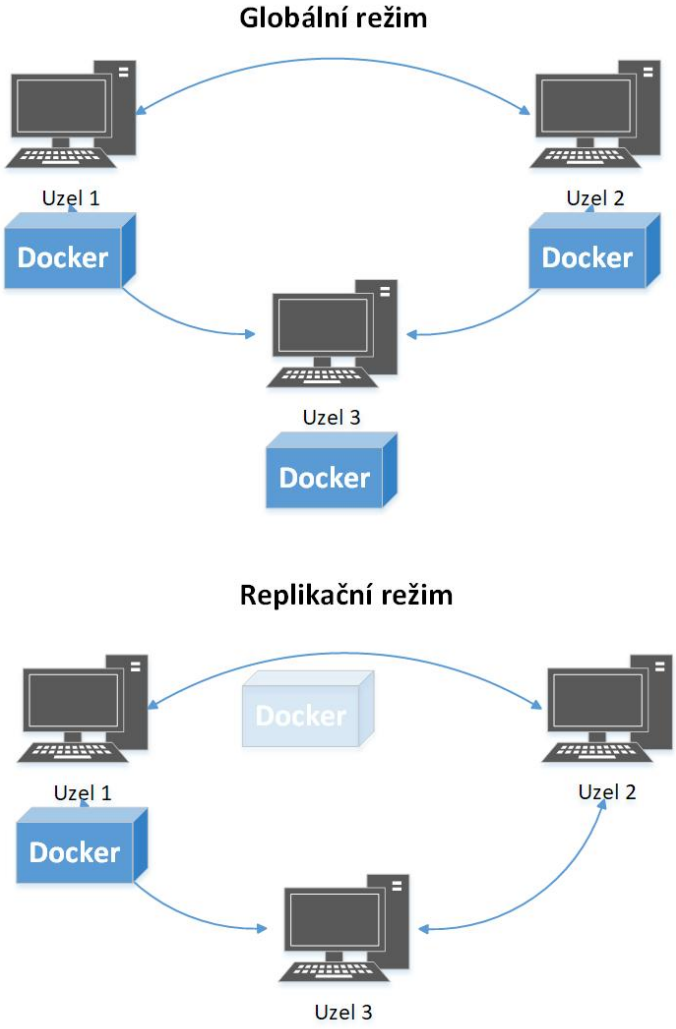
Tabulka 7.5: *Výpis všech uzlů v Docker swarm*

ID	HOSTNAME	STATUS	AVAILABLE	STATUS
cnn4x33hidtwgsrwo2whaksz *	DockerUbuntu1	Ready	Active	Leader
8c1ta4knakuc40t6zapuw0shi	DockerUbuntu2	Ready	Active	Reachable
43jpal45jnj5gwrieoyzyobmp	DockerUbuntu3	Ready	Active	Reachable

Docker definuje dvě role, které můžeme uzlu přidělit, manager a worker. Manager je klíčovým prvkem pro správu a ukládání stavu clusteru. Není dán žádný limit pro počet managerů. Rozhodnutí o tom, kolik manažerských uzlů je implementováno, je kompromis mezi výkonem a odolností proti chybám. Zato role worker slouží pouze k vyvážení zátěže. Danou roli vyčteme z výpisu ze sloupce Manager status. Pokud je pole prázdné znamená to, že je mu přiřazena role worker. Roli uzlů lze měnit a přepínat. V tomto případě mají všechny uzly roli manager. Uzel můžeme přepnout do režimu drain, což znamená, že uzel bude sloužit pouze ke správě, nebude na něj možné spustit žádný kontejner.

7.2.3 Instalace a konfigurace Asterisku

Ústřednu Asterisk můžeme získat dvěma způsoby. Buď ústřednu nainstalovat na svůj operační systém a poté z něj vytvořit vlastní kontejner, nebo použít oficiální zdroje a stáhnout již vytvořený čistý kontejner s ústřednou Asterisk. Tuto možnost jsem využil i já a stáhl Asterisk kontejner ve verzi 13. Nyní zbývá už jen kontejner spustit v režimu swarm. V režimu swarm se vytváří takzvaný servis, který využívá kontejner. Servis můžeme spustit ve dvou režimech, replikovaný a globální. V replikovaném režimu distribuuje správce clusteru mezi uzly určitý počet replikovaných servisů na základě váhy, kterou jste nastavili v požadovaném stavu. Takže servis běží například na jednom z uzlů a v případě výpadku je servis spuštěn na jiném z uzlů, který je dostupný. V globálním režimu je servis spuštěn paralelně na všech uzlech. Aby mohl být servis v tomto režime spuštěn, musí být daný kontejner uložen na každém z uzlů, jinak se servis nespustí a zahlásí chybu.



Obrázek 7.15: *Zobrazení režimu Docker swarm mode*

Servisy se spouštějí a ovládají pomocí předdefinovaných příkazů dockeru. Pro spuštění použijeme tento příkaz. Jednotlivé parametry jsou vysvětleny v tabulce.

```
$ docker service create --name asterisk01 --mode=global --mount
type=bind,source=/swarm/volumes/etc/asterisk,destination=/etc/as
terisk --mount
type=bind,source=/swarm/volumes/var/lib/asterisk,destination=/va
r/lib/asterisk --mount
type=bind,source=/swarm/volumes/var/spool/asterisk,destination=/
var/spool/asterisk --mount
type=bind,source=/swarm/volumes/usr/share/asterisk,destination=/
usr/share/asterisk -p 5060:5060 -p 5060:5060/udp -p 8000-
8100:8000-8100/udp asterisk13-template
```

Tabulka 7.6: *Parametry k spuštění Docker swarm*

Parametr	Vysvětlení
--name	Definice názvu pro servis
--mode	Definice módu, ve kterém bude spuštěn
--mount	Definice externích úložišť hostitele, které lze ke kontejneru připojit
-p	Definice portů, které Asterisk potřebuje pro komunikaci s okolním světem

Nutnou definicí jsou adresáře Asterisku, které obsahují soubory potřebné pro konfiguraci a spuštění. Ty propojíme s úložištěm glusterFS a budou replikovány na všechny nody clusteru. Nadále je zapotřebí definovat porty, pomocí kterých bude Asterisk komunikovat. Pokud toto neprovedeme, nebude možné se k ústředně Asterisk připojit.

Po zadání příkazu je servis spuštěn, v prvních pěti vteřinách je testován, zdali je stabilní a může být propojen se všemi připojenými komponenty. Při úspěšném testování je servis spuštěn a je mu přiděleno identifikační číslo, pomocí kterého je možno se připojit. Všechny důležité informace o servisu lze jednoduše vypsat. Pro připojení použijeme tento příkaz, kde využijeme identifikační číslo.

```
$ docker exec -ti mgs8o5ugwr48 asterisk -rvvvvv
```

Poté můžeme s kontejnerem komunikovat jako s klasickým Asteriskem za pomocí CLI příkazů.

Konfigurační soubory Asterisku byly takřka totožné, jako u předchozího případu a jsou taktéž vloženy v příloze. Ovšem konfigurační soubor *SIP.conf* se lišil v parametru *externip=192.168.1.24*, který bude použit pro zdrojovou IP adresu pro všechny zprávy SIP. V našem případě se bude jednat o zvolenou virtuální IP adresu. Dalším odlišným parametrem je konfigurační soubor *rtp.conf*, jelikož jsme při spuštění definovali komunikační porty, je zapotřebí tyto porty nakonfigurovat i v ústředně Asterisk, viz [28].

Konfigurační soubor obsahuje rozsah portů pro RTP komunikaci, bude vypadat takto:

```
[general]
rtpstart=8000
rtpend=8100
```

Nyní jen ústřednu Asterisk restartujeme, aby mohlo znovu dojít k načtení konfiguračních souborů a konfigurace je u konce.

7.3 Implementace Proxmox

Poslední řešení se zakládá na virtualizaci s použitím živé migrace. K této implementaci jsem použil systém Proxmox Virtual Environment ve verzi 5.1. Jedná se operační systém založeném na open source systému Debian. Aktuálně se vyskytují i jiné systémy se stejným zaměřením, ovšem tento se vyznačuje dobrou uživatelskou podporou a nízkými hardwarovými požadavky. Systém obsahuje webové grafické prostřední a je přizpůsoben k tvorbě virtuálních strojů s využitím KVM virtualizace. Vše bylo testováno na třech virtuálních stojích s níže uvedeným nastavením, každý ze strojů měl vytvořeny dva pevné disky a dvě síťové karty. Jedna sloužila pro komunikaci v clusteru a druhá pro potřeby síťového úložiště Ceph, který je v této implementaci použit.

Tabulka 7.7: *Přehled síťové adresace pro implementaci Proxmox*

Hostname	Popis	IP adresa	Maska	Brána
pve1	Virtuální stroj č.1 - Cluster	158.196.244.134	255.255.255.128	158.196.244.129
	Virtuální stroj č.1 - Ceph	10.10.10.1	255.255.255.0	-
pve2	Virtuální stroj č.2 - Cluster	158.196.244.135	255.255.255.128	158.196.244.129
	Virtuální stroj č.2 - Ceph	10.10.10.2	255.255.255.0	-
pve3	Virtuální stroj č.3 - Cluster	158.196.244.136	255.255.255.128	158.196.244.129
	Virtuální stroj č.3 - Ceph	10.10.10.3	255.255.255.0	-

Po zdárně dokončené instalaci je opět nutné upravit soubor */etc/hosts*, který je vložen v příloze [Příloha H]. Posléze provedeme inicializaci clusteru a to níže uvedeným příkazem na uzlu pve1, *mujcluster* značí pojmenování tohoto clusteru.

```
$ Pvecm create mujcluster
```

Nyní přidáme zbylé dva uzly. To provedeme uvedeným příkazem, který vložíme do uzlů pve2 a pve3. Po vložení je potřeba napsat heslo root, aby uzly provedly autentifikaci a výměnu klíčů a certifikátů pro zabezpečené spojení.

```
$ Pvecm add pve1
```

Pro kontrolu vložíme příkaz *$ Pvecm status*, který vypíše všechny uzly v clusteru.

```
Membership information
----------------------
    Nodeid      Votes Name
0x00000001          1 158.196.244.134 (local)
0x00000002          1 158.196.244.135
0x00000003          1 158.196.244.136
```

Z výpisu vidíme, že všechny uzly jsou přidány do clusteru a mohou navzájem komunikovat. Výpis lze také zobrazit ve webovém grafickém rozhraní.

7.3.1 Ceph

K dosažení vysoké dostupnosti budeme opět využívat sítové úložiště, v tomto případě úložiště Ceph, které je podrobněji popsáno v kapitole 2.3. Toto úložiště je lépe optimalizováno pro práci s většími daty, jako například obraz virtuálního stroje, který bude v této implementaci využit. Hlavní výhodou je, že toto úložiště je integrováno do systému Proxmox VE a k jeho instalaci je potřeba provést jen pár kroků.

Jako první krok nesmíme zapomenout přiřadit IP adresu druhé síťové kartě z rozsahu, který jsme si zvolili. Kompletní nastavení je vloženo v příloze [Příloha I].

Nyní provedeme instalaci aktuální verze úložiště Ceph, pomocí příkazu *$ Pveceph install –version luminous* a to na všech uzlech clusteru. Po dokončení instalace provedeme inicializaci úložiště pro síť, na které bude komunikovat *$ Pveceph init –network 10.10.10.0/24*, to provedeme pouze na uzlu pve1. Na každém uzlu poté musíme vytvořit monitor pomocí *příkazu $ Pveceph createmon*. Ten bude kontrolovat stav úložiště jednotlivých uzlech. Konfigurace je vložena v příloze.

Dalším krokem je vytvoření OSD (Object storage deamon), který je odpovědný za ukládání dat v místním souborovém systému a poskytování přístupu k nim prostřednictvím sítě. Tento krok provedeme přes grafické rozhraní a to tak, že zvolíme daný uzel, v nabídce Ceph vybereme možnost OSD a klikneme na *Create*. Zde zvolíme druhý pevný disk a opět klikneme na *Create*, toto provedeme na každém uzlu [Příloha J]. Instalace síťového úložiště Ceph je nyní hotová.

7.3.2 Asterisk

Instalace a konfigurace je v tomto případě stejná jako je tomu u řešení DRBD, kapitola 7.1.3. Před samostatnou instalací je ovšem nutné vytvořit virtuální stroj v systému Proxmox VE. To povedeme přes nabídku *Create VM* a postupujeme dle instrukcí rozhraní. Ovšem u nabídky Storage zvolíme *pool1_vm*, což je síťové úložiště Ceph. Nyní vytvoříme skupinu HA. Do této skupiny přiřadíme uzly clusteru, mezi kterými se budou v případě výpadku virtuální stroje

přesouvat. Vytvoření skupiny provedeme opět pomocí grafického rozhraní. Vybereme nabídku *Datacentrum*, zde rozklikneme *HA*, zvolíme *Groups* a *Create*. V grafickém okně vybereme uzly, mezi kterými se bude rozkádat vysoká dostupnost. Po dokončení stačí vytvořený VM přiřadit do skupiny pro HA. V případě že uzel, na kterém je VM spuštěn, selže, VM se automaticky přesune na jiný uzel ze skupiny HA. Prioritu jednotlivých uzlů můžeme manuálně upravit. V případě selhání uzlu se nyní VM přesune na jiný, jak můžeme vidět na přiloženém log souboru [Příloha R]. VM se vypne automaticky, přesune se na jiný uzel a znovu se spustí.

V této implementaci jsem pro ústřednu Asterisk zvolil operační systém lubuntu Alternate ve verzi 16.04 z důvodu nižších hardwarových nároků.

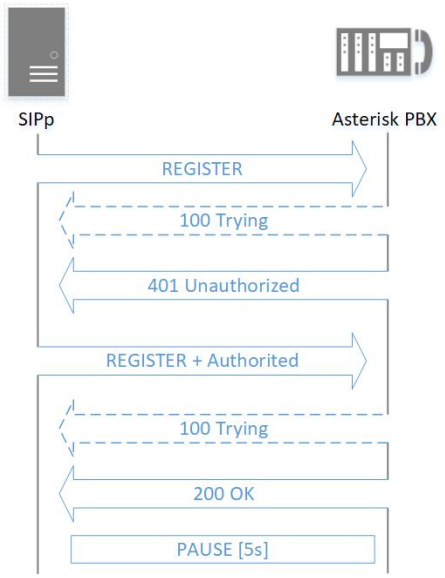
8 Testování

8.1 SIPp

Pro testování jednotlivých implementací jsem zvolil nástroj SIPp. Což je bezplatný testovací nástroj pro protokol SIP. Obsahuje několik základních scénářů, vytváří více hovorů s metodami INVTE a BYE. Dokáže také číst vlastní soubory scénářů XML popisující velmi jednoduché až složité toky hovorů. Obsahuje dynamické zobrazení statistických údajů o běžících testech (frekvence hovoru, zpoždění a statistiku zpráv). Tyto údaje je schopen exportovat do *.csv* souborů, které lze dále zpracovávat, viz [30].

8.2 Scénáře testování

Každá funkční implementace byla prvně otestována pomocí dvou softwarových klientů X-lite a Zoiper, kde byl uskutečněn klasický hovor, aby byla prověřena správná funkčnost. Poté byly implementace testovány nástrojem SIPp, který prováděl dva scénáře. První scénář ověřil registraci a autorizaci klienta. Využíval k tomu definovaný scénář uložený v souboru XML a načítal přihlašovací údaje klientů ze souboru csv.



Obrázek 8.16: *Scénář s požadavkem na registraci*

SIPs v roli klienta zde žádá o registraci vysláním zprávy REGISTER, ústředna automaticky odpovídá informativní zprávou 100 Trying. Jelikož klient není autentizován, vyšle ústředna zprávu 401 Unauthorized obsahující výzvu a metodu, kterou se má autentizace provést. Klient na základně hlasovací funkce zopakuje původní zprávu REGISTER s doplněným polem Authorization obsahující vypočtenou hash. Ústředna opět odpoví informativní zprávou 100 Trying a hash prověří, pokud je vše v pořádku dokončí registraci s odpovědí 200 OK. Následuje pěti sekundová pauza, aby nedošlo k zahlcení.

Tento scénář spustíme následným příkazem:

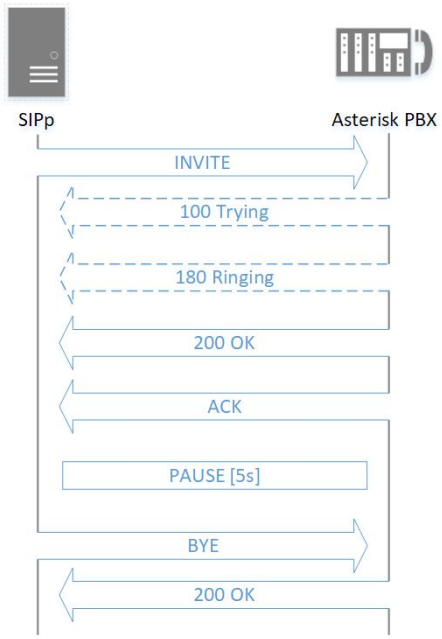
```
./sipp 192.168.1.13 -sf REGISTER_client.xml -inf
REGISTER_client.csv -l 1 -trace_msg -trace_err
```

Tabulka 8.8: *Parametry testovacího scénáře REGISTER*

Parametr	Vysvětlení
-sf	Načtení testovacího scénáře ze zadaného souboru.
-inf	Načtení souboru scv pro vlákání dat.
-l	Počet omezení instancí.
-trace_msg	Zaznamenání zpráv SIP.
-trace_err	Zaznamenání chybových zpráv.

Kompletní scénář i přihlašovací údaje jsou vloženy v příloze [Příloha K, L].

Druhý scénář prověřil sestavení spojení. Při implementaci jsem narážel na spoustu problémů s vytvořením tohoto scénáře. U jednotlivých zpráv docházelo k zacyklení, nebo z neznámých důvodů se klient nemohl spojit s ústřednou. Proto jsem nakonec použil již definovaný scénář s metodou INVIDE, který vyžádal spojení bez registrace a autorizace. Spolu s dial plánem, který byl vytvořen v konfiguračním souboru ústředny *extensions.conf*.



Obrázek 8.17: *Scénář s požadavkem na spojení*

SIPp zde žádá o spojení a vyšle tak požadavek INVITE, ústředna automaticky odpoví informativní zprávou 100 Trying a 180 Ringing, která oznamuje vyzvánění. Hovor je přijat s odpovědí 200 OK. Ústředna dále reaguje potvrzující zprávou ACK. Nyní nastane pěti sekundová pauza, která simuluje hovor. Po pěti sekundách SIPp požádá o ukončení a vyšle zprávu BYE. Ústředna reaguje zprávou 200 OK.

Scénář spustíme následujícím příkazem:

```
./sipp -sn uac -d 10000 -s 1002 192.168.1.13 -l 3
```

Tabulka 8.9: *Parametry testovacího scénáře INVITE*

Parametr	Vysvětlení
-sn	Parametr pro použití vnořeného scénáře
-d	Zpoždění v milisekundách
-l	Počet omezení instancí.
-s	Nastaví uživatelskou část URI požadavku.

V tomto scénáři byly vyslány vždy tři iterace najednou, z důvodu vetší pravděpodobnosti daného požadavku a také pro otestování stability.

8.3 Naměřené hodnoty

Pro každou funkční implementaci vysoké dostupnosti byly otestovány oba scénáře. Měření započalo spuštěním testu. V průběhu testování byla ústředna úmyslně odpojena a byl měřen čas, a také počet správných a špatných provedených iterací, dokud záložní ústředna plně nepřevzala svou úlohu. Jednotlivé scénáře probíhaly stále dokola, dokud nebyl test manuálně ukončen. Nástroj SIPp vyčkal, než proběhne aktuální iterace dokonce, poté test ukončil a zaznamenal naměřené hodnoty. Jakmile byla během testování vyslána žádost na ústřednu, SIPs vyčkával na odpověď. Pokud odpověď nedorazila, opakoval vyslání žádosti. Po překonání stanovených limitů pro odeslání žádosti byla iterace vyhodnocena jako neúspěšná a scénář započal od začátku. Každý scénář byl otestován celkem třicetkrát, z naměřených hodnot byla poté provedena analýza a vyhotoveny grafy. Ukázkový výsledek testování je vložen v příloze [Příloha M].

Testováno bylo řešení vysoké dostupnosti s replikačním úložištěm DRBD spojené se systémem sledování dostupnosti Pacemakerem. Dále řešení na bázi kontejneru a systému Docker swam v globálním režimu. Avšak při testování systému Docker v replikačním režimu docházelo k zacyklení jednotlivých požadavků a test byl zdárně dokončen pouze třikrát, v jiných případech test vyhodnocoval špatné iterace, i když záložní ústředna byla plně přístupná pro softwarové klienty. Poslední řešení založené na živé migraci a systému Proxmox se bohužel nepovedlo z technických důvodů uskutečnit. Vyskytly se problémy se síťováním, dále byl systém velmi háklivý na výkon hardwaru. Spuštění virtuálního stoje probíhalo v některých případech i v rámci minut, což by mělo veliký vliv na výsledky testování. Vyskytly se případy, kdy stroj nešel spustil vůbec.

Všechny naměřené hodnoty jsou vloženy v příloze. U každého z měření byla zaznamenána celková doba, po kterou test probíhal. Dále počet špatně, dobře a celých počet provedených iterací a průměrná doba, za kterou jedna iterace proběhla. Z těchto hodnot bylo možno vypočíst dobu nečinnosti, po kterou byla ústředna Asterisk nedostupná.

Všechny naměřené hodnoty jsou zaznamenány v tabulkách a vloženy v příloze. Řešení DRBD a Pacemaker s testovacím scénářem pro metodu INVITE [Příloha N], pro metodu REGISTER [Příloha O]. Řešení na bázi systému Docker swarm s testovacím scénářem pro metodu INVITE [Příloha P], pro metodu REGISTER [Příloha Q]. Ze všech hodnot byly průměrné, maximální, minimální hodnoty a vloženy do tabulky.

Tabulka 8.10: *Výsledné hodnoty špatně přijatých iterací pro scénář INVITE*

	Implementace DRBD a Pacemaker	Implementace Docker swam – Global mode
Medián	7	3
Průměr	7	6
Minimum	3	0
Maximum	9	56

Tabulka 8.11: *Výsledná doba nečinnosti pro scénář INVITE*

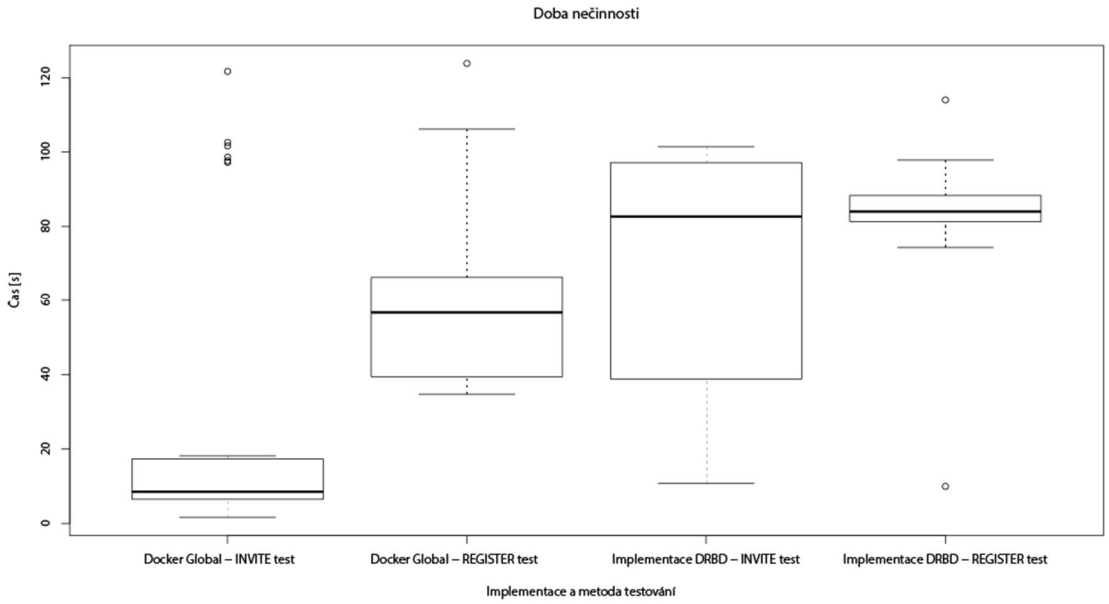
	Implementace DRBD a Pacemaker	Implementace Docker swam – Global mode
Medián [s]	82.721	8.530
Průměr [s]	67.684	27.325
Minimum [s]	10.868	1.706
Maximum [s]	101.458	121.750

Tabulka 8.12: *Výsledné hodnoty špatně přijatých iterací pro scénář REGISTER*

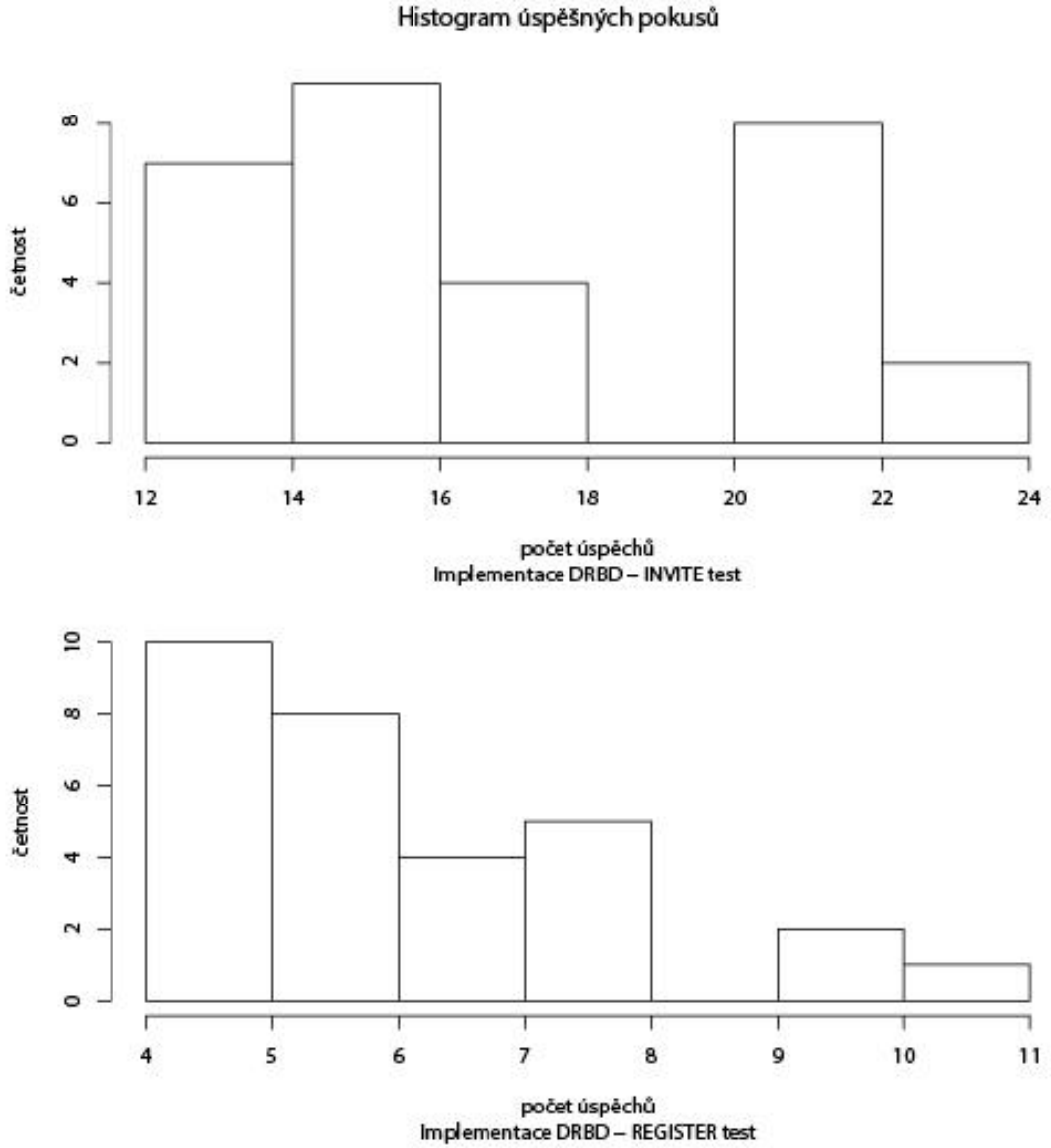
	Implementace DRBD a Pacemaker	Implementace Docker swam – Global mode
Medián	4	7
Průměr	4	7
Minimum	3	3
Maximum	8	12

Tabulka 8.13: *Výsledná doba nečinnosti pro scénář REGISTER*

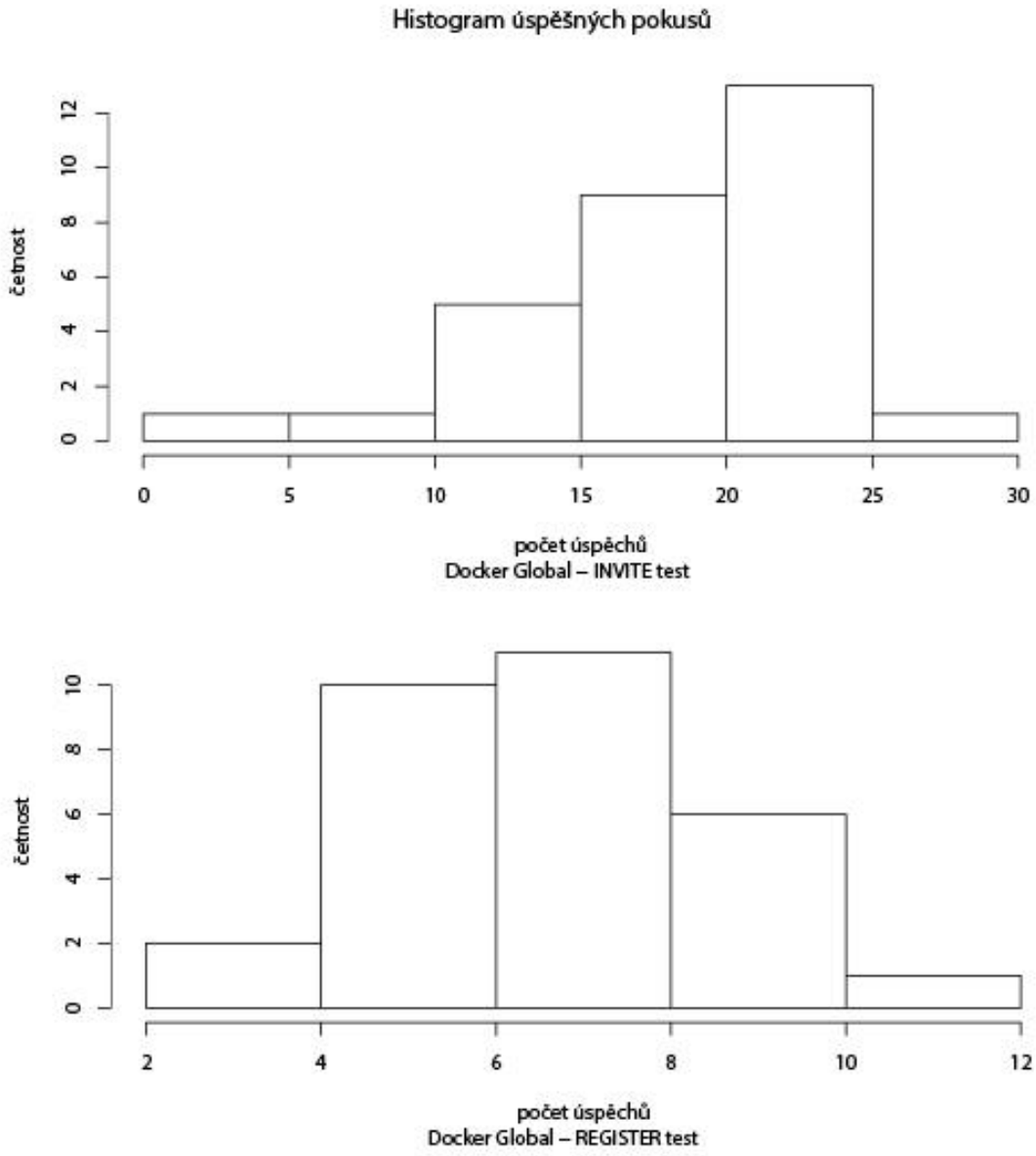
	Implementace DRBD a Pacemaker	Implementace Docker swam – Global mode
Medián [s]	84.041	56.708
Průměr [s]	83.376	57.185
Minimum [s]	9.994	34.713
Maximum [s]	114.053	123.885



Obrázek 8.18: *Krabicový graf znázorňující dobu nečinnosti pro všechna řešení a testovací scénáře*



Obrázek 8.19: *Histogram znázorňující počet úspěšných iterací pro implementaci DRBD a Pacemaker*



Obrázek 8.20*: Histogram znázorňující počet úspěšných iterací pro implementaci Docker swarm*

Z přiložených histogramů, které znázorňují počet úspěšných iterací, lze vysledovat, že hodnoty nejsou normálně rozdělené. Proto není možná další statistická analýza.

Závěr

Cílem mé diplomové práce bylo zajistit vysokou dostupnost pro SIP server za použití dnešních technologií. Aktuálně se nabízí veliká škála řešení, jak vysokou dostupnost zajistit. Nabízejí se komerční produkty, které jsou zaměřeny jen na tento typ problému. Řešení mohou být jak hardwarového, tak softwarového typu. Ovšem já se touto prací zaměřil na volně dostupná softwarová řešení, která se aktuálně nabízí. Po prozkoumání možností, jsem zjistil, že řešení jak zajistit vysokou dostupnost, je opravdu mnoho a byl problém vybrat ty nejvíce vyhovující. Nakonec jsem zvolil celkem tři řešení, každé s velmi odlišnou technologií, abych otestoval jejich výhody a nevýhody. Také jsem chtěl zjistit jejich funkčnost se SIP server Asterisk. Během realizací, které jsem prováděl ve virtualizovaném prostředí, jsem průběžně objevoval nedokonalosti technologií, které jsem použil a musel jsem vynalézt způsob, jak je opravit, či obejít.

První řešení využívalo replikaci dat mezi dvěma stanicemi s použitím síťového úložiště DRBD a výměny virtuální IP adresy, která byla vždy přiřazena hlavnímu uzlu. Tento uzel sloužil k přihlašování klientů. Ústředna Asterisk byla spuštěna na obou stanicích, které využívaly stejné úložiště, v případě výpadku se virtuální IP adresa přesunula na záložní uzel. Přesun IP adresy trval v řádu milisekund, pouhé restartování síťových služeb zapříčinilo přesun adresy na záložní stanici. Na stav stanic dohlížel systém Pacemaker, jenž nabízí veliké množství možností, co se týče konfigurace. Dokáže nejen přesouvat IP adresy, ale také měnit režimy, či spouštět aplikace.

Pro druhé řešení jsem použil technologií kontejnerů se systémem Docker. Kontejnery se dnes začínají hojně rozšiřovat, proto jsem chtěl prověřit jejich uplatnění ve VoIP. Samotný Docker nabízí možnosti, jak zajistit vysokou dostupnost. Ovšem má i své nevýhody, nedokáže pevně přidělit IP adresu danému kontejneru. Proto jsem zde opět využil služeb Pacemakeru a virtuální IP adresy, která funguje výborně. Kontejnery mají i další omezení, například práce s porty, jež se musí pevně definovat, což může být omezující pro hromadné nasazení. Ale dle oficiálních zdrojů se pracuje na nápravě a problémy by měly být časem odstraněny. Kontejnery mají ale také řadu výhod. Aplikace, které jsou v nich zabaleny, lze velmi snadno přesouvat a jejich velikost, i zátěž na hardware je velmi malá.

Poslední řešení využívá virtualizaci strojů a živou migraci. Tato implementace obsahovala takřka všechny prostředky a nebylo nutné ji doplňovat o další technologie. Nicméně nepovedlo se mi ji z technických důvodů realizovat spolu s ústřednou Asterisk. Technologie je velmi háklivá na výkon hardwaru. Při testování vysoké dostupnosti živá migrace trvala v průměru 2,6s. Ovšem při testování výpadku uzlu trvalo delší dobu, než na něj systém zareagoval zhruba 66s, k tomu je nutno připočíst čas, který je potřebný k nastartování virtuálního stroje, což mém případě bylo i v rámci několika minut.

Nutno podotknout, že ústředna využívala výchozí databázi z důvodu testování jejího vlivu při vysoké dostupnosti. Samostatné zareagování na výpadek umí testované technologie v rámci milisekund, až sekund. Za delší časy může v tomto případě samotná ústředna Asterisk, která má své limity. Další možností je použití jiných databází, které jsou na vysokou dostupnost lépe

přizpůsobeny. Testování probíhalo pomocí analyzátoru SIPp, všechny výsledky jsou zde uvedeny v tabulkách a grafech. Z hodnot vyplývá, že nejlépe je na tom řešení se systémem Docker, ovšem pro své omezení jej doporučuji pouze pro testování, či menší firmy.

Použitá literatura

[1] Travnicek, L. Komunikační server s vysokou dostupností.VŠB-TU Ostrava, Diplomová Práce, 2010.

[2] Hlavacek, J. Robustness of VoIP systems, FEL ČVUT v Praze, Dizertační práce 2015.

[3] Meggelen, J., Smith, J., Madsen, L., Asterisk: The Future of Telephony. O'Reilly, 2007

[4] Voznak, M., Voice over IP. VSB-Technical University of Ostrava:, 2008

[5] WINTERMEZER, S.; BOSCH, S. Practical Asterisk 1.4 and 1.6: From Beginner to Expert. Addison-Wesley Professional; 1 edition, 2009. ISBN 978-0-321-52566-6

[6] SINNREICH, H., JOHNSTON, A. B. Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol. Canada: Wiley, 2006. ISBN 978-0471776574

[7] Kanso, A., Lemieux, Y., "Achieving High Availability at the Application Level in the Cloud," 2013 IEEE Sixth International Conference on Cloud Computing, 2013, pp. 778-785

[8] Hlaváček, J., Bešťák, R., "Improvements in the Availability of SIP Networks," In Proceedings of the 2010 Networking and Electronic Commerce Research Conference. Dallas, TX: American Telecommunications Systems Management Association Inc., 2010, pp. 109-117

[9] Koutras, V. P., Platis, A. N., "VoIP Availability and Service Reliability through Software rejuvenation policies," In Proceedings of the 2nd International Conference on Dependability of Computer Systems, 2nd International Conference on Dependability of Computer Systems, 2007, pp. 262–269.

[10] Bicom Systems [online], URL: <https://www.bicomsystems.com/products/serverware>

[11] Asterisk High Availability Design [online], <URL: https://www.voip-info.org/asterisk-high-availability-design>

[12] DRBD [online], URL: <http://www.drbd.org/>

[13] Thirdlane [online], URL: <https://www.thirdlane.com/products/uc-cluster>

[14] FreeBPX [online], URL: <https://www.freepbx.org/high-availability/>

[15] Elestix [online], URL: <https://www.elastix.org/blog/elastixworld/>

[16] High Availability for SARK [online], URL: <http://www.sailpbx.com/mediawiki/>

[17] Thirdlane [online], URL: <https://www.thirdlane.com/products/uc-cluster>

[18] DRBD: a distributed block device [online], URL: <https://lwn.net/Articles/329543>

[19] GlusterFS [online], URL: <https://redhatstorage.redhat.com/products/glusterfs/>

[20] GlusterFS – Geo-replication v praxi, Libor Zoubek [online], URL: <https://www.linuxexpres.cz/software/glusterfs-geo-replication-v-praxi>

[21] Proxmox [online], URL: <https://www.proxmox.com/>

[22] What is Proxmox?, Javier Vidueira [online], URL: <https://www.hivelocity.net/kb/what-is-proxmox/>

[23] Sphere HA [online], URL: <https://pubs.vmware.com/>

[24] oVirt [online], URL: <https://www.ovirt.org/>

[25] SIP [online], URL: <https://www.voip-info.org/sip>

[26] Asterisk PBX, Cervenka [online], URL: <https://sip.cesnet.cz/cs/swahw/asterisk>

[27] Virtualization [online], URL: <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>

[28] Docker [online], URL: <https://www.docker.com/>

[29] Docker a Windows, Michal Augustý [online], URL: <https://www.zdrojak.cz/clanky/docker-a-windows/>

[30] SIPp [online], URL: <http://sipp.sourceforge.net>

[31] Cepp [online], URL: <http://docs.ceph.com/>

[32] SIPP stress test tool, Venkatesh Macha [online], URL: <https://voip4learn.blogspot.cz/2015/09/sipp-installation-and-testing-asterisk.html>

[33] O2BC Cluster [online], URL: <https://www.lisenet.com/2016/o2cb-cluster-with-dual-primary-drbd-and-ocfs2-on-oracle-linux-7/>

[34] Asterisk on Docker, George Lucian Tabacar [online], URL: <https://www.purplesrl.com/en/making-calls-using-asterisk-docker/>

[35] Cluster na Linuxu: vysoká dostupnost s RHEL a deriváty, Ondřej Beneš [online], URL: < https://www.root.cz/clanky/cluster-na-linuxu-vysoka-dostupnost-s-rhel-a-derivaty/>

Seznam příloh

Příloha A: *Konfigurační soubor corosync.conf*

```
totem {
  version: 2
  cluster_name: lbcluster
  transport: udpu
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.1.13
    broadcast: yes
    mcastport: 5405
  }
}
quorum {
  provider: corosync_votequorum
  two_node: 1
}
nodelist {
  node {
    ring0_addr: 192.168.1.11
    name: primary
    nodeid: 1
  }
  node {
    ring0_addr: 192.168.1.12
    name: secondary
    nodeid: 2
  }
}
logging {
  to_logfile: yes
  logfile: /var/log/corosync/corosync.log
```

```
  to_syslog: yes
  timestamp: on
}
```

Příloha B: *Konfigurační soubor datastore.res*

```
resource datastore {
        Logical common/shared device for storage
        device  /dev/drbd0;
        # meta information
        meta-disk internal;
        # disk address
        disk /dev/sdb1;
        startup { become-primary-on both; }

        net {
                allow-two-primaries yes;
                after-sb-0pri discard-zero-changes;
                after-sb-1pri discard-secondary;
                after-sb-2pri disconnect;
        }
        # define node 1 information
        on node01 { address 192.168.1.11:7789; }
        # define node 2 information
        on node02 { address 192.168.1.12:7789; }
```

Příloha C: *Konfigurační soubor cluster.conf*

```
node:
    ip_port = 7777
    ip_address = 192.168.1.11
    number = 0
    name = Ubuntu1
```

```
    cluster = ocfs2
node:
    ip_port = 7777
    ip_address = 192.168.1.12
    number = 1
    name = Ubuntu2
    cluster = ocfs2
cluster:
    node_count = 2
    name = ocfs2
```

Příloha D: *Konfigurační soubor sip.conf*

```
[general]
context=internal
allowguest=no
allowoverlap=no
bindport=5060
bindaddr=192.168.1.13
srvlookup=no
disallow=all
allow=ulaw
alwaysauthreject=yes
canreinvite=no
nat=yes
session-timers=refuse
localnet=192.168.1.0/255.255.255.0

[1000]
type=friend
language=cz
host=dynamic
```

```
secret=1000
context=internal

[2000]
type=friend
language=cz
host=dynamic
secret=2000
context=internal

[sipp]
type=friend
context=sipp
host=dynamic
port=6000
user=sipp
```

Příloha E: *Konfigurační soubor extensions.conf*

```
CONSOLE=Console/dsp
[demo]
include => stdexten
exten => s,1,Wait(1)
exten => s,n,Answer
exten => s,n,Set(TIMEOUT(digit)=5)
exten => s,n,Set(TIMEOUT(response)=10)
exten => s,n(restart),BackGround(demo-congrats)
exten => s,n(instruct),BackGround(demo-instruct)
exten => s,n,WaitExten

exten => 2,1,BackGround(demo-moreinfo)
exten => 2,n,Goto(s,instruct)
```

```
exten => 3,1,Set(CHANNEL(language)=fr)
exten => 3,n,Goto(s,restart)
exten => 1000,1,Goto(default,s,1)
[internal]
exten => _X.,1,Dial(SIP/${EXTEN},30)
exten => _X.,2,hangup()

[sipp]
exten => 1001,1,Answer
exten => 1001,n,SetMusicOnHold(default)
exten => 1001,n,WaitMusicOnHold(20)
exten => 1001,n,Hangup
exten => 1002,1,Answer
exten => 1002,n,Goto(demo,s,1)
exten => 1002,n,Hangup
```

Příloha F: *Konfigurační soubor asterick.conf*

```
[directories]
astetcdir => /home/daniel/Asterisk/etc/asterisk
astmoddir => /home/daniel/Asterisk/usr/lib/asterisk/modules
astvarlibdir => /home/daniel/Asterisk/var/lib/asterisk
astdbdir => /home/daniel/Asterisk/var/lib/asterisk
astkeydir => /home/daniel/Asterisk/var/lib/asterisk
astdatadir => /home/daniel/Asterisk/var/lib/asterisk
astagidir => /home/daniel/Asterisk/var/lib/asterisk/agi-bin
astspooldir => /home/daniel/Asterisk/var/spool/asterisk
astrundir => /var/run/asterisk
astlogdir => /home/daniel/Asterisk/var/log/asterisk
astsbindir => /usr/sbin
```

Příloha G: *Konfigurační soubor hosts pro implementaci Docker*

```
127.0.0.1       localhost
127.0.1.1       DockerUbuntu1
192.168.1.21    DockerUbuntu1
192.168.1.22    DockerUbuntu2
192.168.1.23    DockerUbuntu3
```

Příloha H: *Konfigurační soubor hosts pro implementaci Proxmox*

```
127.0.0.1 localhost.localdomain localhost
158.196.244.134 pve1.vsb.cz pve1 pvelocalhost
158.196.244.135 pve2.vsb.cz pve2
158.196.244.136 pve3.vsb.cz pve3
```

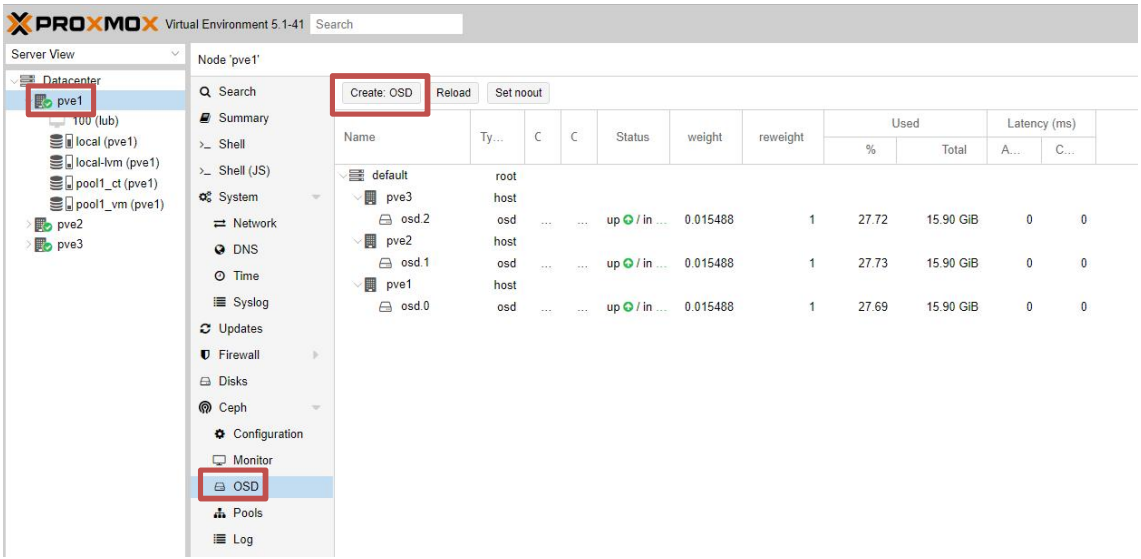
Příloha I: *Konfigurace síťových rozhraní pro implementaci Proxmox*

```
auto lo
iface lo inet loopback
iface ens192 inet manual

auto ens224
iface ens224 inet static
address  10.10.10.1
netmask  255.255.255.0

auto vmbr0
iface vmbr0 inet static
address  158.196.244.134
netmask  255.255.255.128
gateway  158.196.244.129
bridge_ports ens192
bridge_stp off
bridge_fd 0
```

Příloha J: *Vytvoření OSD v grafickém rozhraní*



Příloha K: *Testovací scénář pro registraci klienta*

```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<scenario name="register_client">
  <send retrans="500">
    <![CDATA[

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From: <sip:[field0]@[field1]>;tag=[call_number]
      To: <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: [cseq] REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      Max-Forwards: 10
      Expires: 120
      User-Agent: SIPp/Win32
      Content-Length: 0
```

```
    ]]>
  </send>
  <!-- asterisk -->
  <recv response="100" optional="true">
  </recv>
  <recv response="401" auth="true">
  </recv>
  <send retrans="500">
    <![CDATA[
      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From: <sip:[field0]@[field1]>;tag=[call_number]
      To: <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: [cseq] REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      [field2]
      Max-Forwards: 10
      Expires: 120
      User-Agent: SIPp/Win32
      Content-Length: 0
    ]]>
  </send>
  <!-- asterisk -->
  <recv response="100" optional="true">
  </recv>
  <recv response="200">
  </recv>
  <pause milliseconds="5000"/>
  <!-- response time repartition table (ms)   -->
```

```
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150,
200"/>
  <!-- call length repartition table (ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000,
10000"/>
</scenario>
```

Příloha L: *Přihlašovací údaje*

```
SEQUENTIAL
1000;192.168.1.13;[authentication username=1000
password=1000];2000;
2000;192.168.1.13;[authentication username=2000
password=2000];1000;
```

Příloha M: *Ukázkový výsledek testování*

```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
10.0(5000 ms)/1.000s   5060     109.40 s           21  192.168.1.13:5060(UDP)

  0 new calls during 0.000 s period      0 ms scheduler resolution
  0 calls (limit 3)                      Peak was 3 calls, after 0 s
  0 Running, 13 Paused, 0 Woken up
  0 dead call msg (discarded)            0 out-of-call msg (discarded)
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->          21        30        3
        100 <----------          18        0         0         0
        180 <----------          0         0         0         0
        183 <----------          0         0         0         0
        200 <----------  E-RTD1  18        0         0         0
        ACK ---------->          18        0
      Pause [   5000ms]          18                            0
        BYE ---------->          18        27        3
        200 <----------          15        0         0         0

------------------------------ Test Terminated --------------------------------


------------------------------ Statistics Screen ------- [1-9]: Change Screen --
  Start Time                | 2018-04-17 00:03:30.804194 1523916210.804194
  Last Reset Time           | 2018-04-17 00:05:20.222011 1523916320.222011
  Current Time              | 2018-04-17 00:05:20.223854 1523916320.223854
--------------------------+---------------------------+--------------------------
  Counter Name              | Periodic value            | Cumulative value
--------------------------+---------------------------+--------------------------
  Elapsed Time              | 00:00:00:001000           | 00:01:49:419000
  Call Rate                 |    0.000 cps              |    0.192 cps
--------------------------+---------------------------+--------------------------
  Incoming call created     |        0                  |        0
  OutGoing call created     |        0                  |       21
  Total Call created        |                           |       21
  Current Call              |        0                  |
--------------------------+---------------------------+--------------------------
  Successful call           |        0                  |       15
  Failed call               |        0                  |        6
--------------------------+---------------------------+--------------------------
  Response Time 1           | 00:00:00:000000           | 00:00:02:590000
  Call Length               | 00:00:00:000000           | 00:00:15:520000
------------------------------ Test Terminated --------------------------------
```

Příloha N: *Tabulka výsledných hodnot pro implementaci DRBD a Pacemaker – testování scénáře INVITE*

Měření	Délka měření [s]	Celkový počet iterací	Počet správných iterací	Počet špatných iterací	Průměrná doba jedné iterace	Doba nečinnosti [s]
1	48.248	24	21	3	5.34	10.868
2	53.620	27	24	3	5.34	10.900
3	48.514	24	21	3	5.34	11.134
4	51.812	23	16	7	5.34	23.332
5	103.001	27	21	6	5.34	65.621
6	111.623	21	12	9	5.34	90.263
7	60.214	21	12	9	5.34	38.854
8	109.420	21	15	6	5.34	82.720
9	59.617	27	21	6	5.34	22.237
10	125.519	24	15	9	5.34	98.819
11	140.618	31	22	9	5.34	101.458
12	106.720	25	16	9	5.34	78.240
13	130.617	27	18	9	5.34	98.577
14	120.424	21	12	9	5.34	99.064
15	109.421	21	15	6	5.34	82.721
16	66.546	24	15	9	5.34	39.846
17	116.712	30	24	6	5.34	73.992
18	120.422	21	12	9	5.34	99.062
19	55.000	24	18	6	5.34	22.960
20	118.589	21	12	9	5.34	97.229
21	109.000	24	21	3	5.34	71.620
22	109.584	21	15	6	5.34	82.884
23	109.251	24	21	3	5.34	71.871
24	113.584	21	12	9	5.34	92.224
25	110.147	21	15	6	5.34	83.447
26	48.514	24	21	3	5.34	11.134
27	112.000	23	16	7	5.34	83.520
28	129.258	27	18	9	5.34	97.218
29	112.558	21	12	9	5.34	91.198
30	129.547	27	18	9	5.34	97.507

Tabulka výsledných hodnot pro implementaci DRBD a Pacemaker – testování scénáře REGISTER

Příloha O: *Tabulka výsledných hodnot pro implementaci DRBD a Pacemaker – testování scénáře REGISTER*

Měření	Délka měření [s]	Celkový počet iterací	Počet správných iterací	Počet špatných iterací	Průměrná doba jedné iterace	Doba nečinnosti [s]
1	67.117	19	11	8	5.193	9.994
2	135.911	14	8	6	5.193	94.367
3	129.874	14	8	6	5.193	88.330
4	139.411	13	8	5	5.193	97.867
5	115.458	10	6	4	5.193	84.300
6	113.513	13	7	6	5.193	77.162
7	113.516	10	6	4	5.193	82.358
8	121.513	10	6	4	5.193	90.355
9	115.313	8	4	4	5.193	94.541
10	121.518	10	6	4	5.193	90.360
11	123.955	13	8	5	5.193	82.411
12	140.018	10	5	5	5.193	114.053
13	138.118	16	10	6	5.193	86.188
14	108.316	8	5	3	5.193	82.351
15	108.312	8	5	3	5.193	82.347
16	112.333	8	5	3	5.193	86.368
17	100.314	8	5	3	5.193	74.349
18	99.216	7	4	3	5.193	78.444
19	117.518	10	6	4	5.193	86.360
20	114.618	11	7	4	5.193	78.267
21	116.333	13	7	6	5.193	79.982
22	114.516	10	6	4	5.193	83.358
23	109.746	8	5	3	5.193	83.781
24	134.422	13	8	5	5.193	92.878
25	136.654	16	10	6	5.193	84.724
26	107.586	8	5	3	5.193	81.621
27	115.647	10	6	4	5.193	84.489
28	117.413	13	7	6	5.193	81.062
29	107.316	8	5	3	5.193	81.351
30	118.416	10	6	4	5.193	87.258

Tabulka výsledných hodnot pro implementaci Docker swarm v režimu global – testování scénáře INVITE

Příloha P: *Tabulka výsledných hodnot pro implementaci Docker swarm v režimu global – testování scénáře INVITE*

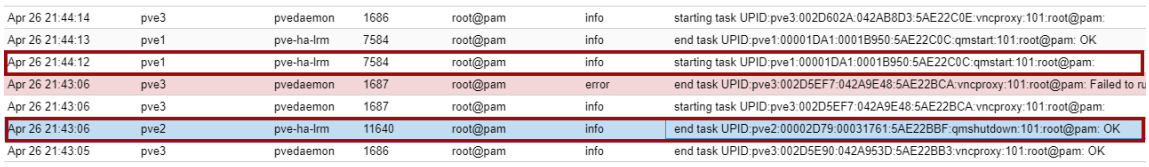
Měření	Délka měření [s]	Celkový počet iterací	Počet správných iterací	Počet špatných iterací	Průměrná doba jedné iterace	Doba nečinnosti [s]
1	40.515	18	18	0	5.34	8.475
2	39.621	19	16	3	5.34	11.141
3	48.513	25	23	2	5.34	7.573
4	44.625	24	21	3	5.34	7.245
5	55.512	23	21	2	5.34	18.132
6	33.220	15	15	0	5.34	6.520
7	127.090	15	3	9	5.34	121.750
8	44.426	24	24	0	5.34	1.706
9	129.333	53	15	38	5.34	102.633
10	133.726	21	18	9	5.34	101.686
11	29.327	15	15	0	5.34	2.627
12	151.126	39	30	9	5.34	97.726
13	44.532	24	21	3	5.34	7.152
14	43.630	21	18	3	5.34	11.590
15	47.927	27	24	3	5.34	5.207
16	130.724	27	18	9	5.34	98.684
17	33.220	15	15	0	5.34	6.520
18	43.925	21	21	0	5.34	6.545
19	38.326	18	18	0	5.34	6.286
20	113.316	56	9	56	5.34	97.296
21	43.730	21	18	3	5.34	11.690
22	48.337	27	24	3	5.34	5.617
23	40.681	19	16	3	5.34	12.201
24	49.587	27	24	3	5.34	6.867
25	33.220	15	15	0	5.34	6.520
26	45.965	24	21	3	5.34	8.585
27	54.692	23	21	2	5.34	17.312
28	45.426	24	24	0	5.34	2.706
29	39.361	19	16	3	5.34	10.881
30	48.254	24	21	3	5.34	10.874

Tabulka výsledných hodnot pro implementaci Docker swarm v režimu global – testování scénáře REGISTER

Příloha Q: *Tabulka výsledných hodnot pro implementaci Docker swarm v režimu global – testování scénáře REGISTER*

Měření	Délka měření [s]	Celkový počet iterací	Počet správných iterací	Počet špatných iterací	Průměrná doba jedné iterace	Doba nečinnosti [s]
1	78.694	13	8	5	5.193	37.150
2	76.619	13	8	5	5.193	35.075
3	68.368	11	6	5	5.193	37.210
4	82.613	11	6	5	5.193	51.455
5	113.512	13	8	5	5.193	71.968
6	81.450	14	9	5	5.193	34.713
7	84.719	14	7	7	5.193	48.368
8	129.684	19	12	7	5.193	67.368
9	98.771	15	7	8	5.193	62.420
10	102.821	16	9	7	5.193	56.084
11	112.000	14	10	4	5.193	60.070
12	139.464	8	3	5	5.193	123.885
13	72.039	10	6	4	5.193	40.881
14	88.015	11	7	4	5.193	51.664
15	99.296	13	5	8	5.193	73.331
16	73.015	11	7	4	5.193	36.664
17	88.490	11	6	5	5.193	57.332
18	121.768	11	3	7	5.193	106.189
19	83.320	11	6	5	5.193	52.162
20	101.882	10	6	4	5.193	70.724
21	104.658	16	9	7	5.193	57.921
22	70.569	11	6	5	5.193	39.411
23	68.147	10	6	4	5.193	36.989
24	114.154	14	9	5	5.193	67.417
25	104.296	13	8	5	5.193	62.752
26	69.968	11	6	5	5.193	38.810
27	99.654	14	8	6	5.193	58.110
28	102.657	15	7	8	5.193	66.306
29	105.695	13	8	5	5.193	64.151
30	95.695	14	9	5	5.193	48.958

Příloha R: *Logovací soubor ze systému Proxmox při testování HA*



Apr 26 21:44:14	pve3	pvedaemon	1686	root@pam	info	starting task UPID:pve3:002D602A:042AB8D3:5AE22C0E:vncproxy:101:root@pam:
Apr 26 21:44:13	pve1	pve-ha-lrm	7584	root@pam	info	end task UPID:pve1:00001DA1:0001B950:5AE22C0C:qmstart:101:root@pam: OK
Apr 26 21:44:12	pve1	pve-ha-lrm	7584	root@pam	info	starting task UPID:pve1:00001DA1:0001B950:5AE22C0C:qmstart:101:root@pam:
Apr 26 21:43:06	pve3	pvedaemon	1687	root@pam	error	end task UPID:pve3:002D5EF7:042A9E48:5AE22BCA:vncproxy:101:root@pam: Failed to ru
Apr 26 21:43:06	pve3	pvedaemon	1687	root@pam	info	starting task UPID:pve3:002D5EF7:042A9E48:5AE22BCA:vncproxy:101:root@pam:
Apr 26 21:43:06	pve2	pve-ha-lrm	11640	root@pam	info	end task UPID:pve2:00002D79:00031761:5AE22BBF:qmshutdown:101:root@pam: OK
Apr 26 21:43:05	pve3	pvedaemon	1686	root@pam	info	end task UPID:pve3:002D5E90:042A953D:5AE22BB3:vncproxy:101:root@pam: OK